Software Engineering for the Research Environment

A process of improving Process Improvement Processes in academic research

Andre Shem Oboler

B.Comp.Sci.(Hons)

Computing Department
Lancaster University

Presented for the degree of Doctor of Philosophy at Lancaster University, UK

July 2007
Abstract
This thesis presents a new process based approach to software engineering designed to meet the needs of academic computer science researchers. The core objective was to examine whether software engineering approaches could be adapted for the research environment so that they gain acceptance and enable improvement of the research process.

The approach included the provision of selected process descriptors (software, tools, and guidelines) to support research. Inter-researchers support, and support between researchers and a departmental Software Engineer were examined. A new working paradigm for computer science departments was simulated in our experiments; this involves the integration of a software engineer working on projects across the department. The approach encourages researcher’s reflection and conscious engagement with their research process.

For researchers, our approach provides novel ways of documenting the research and research process and sharing this information in a low cost manner. Selected software engineering tools and approaches were evaluated for both their benefits and acceptability to the academic community. A framework for judging academic capability maturity is presented.

Five-month research projects by MSc students form the experimental basis. Data was collected over three years. About half the students opted for the experimental approach. Data collection ranged from inspection of outputs, interviews, and surveys to software engineering approaches like Formal Technical Reviews. In a case study approach we present information on the attitudes, approaches and rationale of the researchers, such understanding is needed for new approaches to be both helpful and acceptable.
Developed in an evolutionary manner, changes and the rationale for them are explained, demonstrating a growing understanding of the needs of researchers and of the research environment.

This work provides a foundation for an area of growing interest in computer science research, and a means of increasing the rate of scientific progress across the Science and Technology domains.
Declaration

I declare that this thesis is my own work and has not been submitted in substantially the same form for the award of a higher degree elsewhere. Publications arising from this work have been listed. While the initial concept for this work is partly based on ideas from the USE CSR project, conducted by myself for a previous degree, this thesis presents a new approach known as SERE and has extended far beyond the initial concept.

This thesis is © Andre Oboler 2007, and permission is hereby given for the abstract, table of contents and introduction to be copied and distributed in physical or electronic form upon request for the purposes of academic research or private study. The full thesis may be made available according to the standard practices of Lancaster University (as they regard thesis), including both local access and interlibrary loans.

Andre Oboler
July, 2007
Lancaster University
Acknowledgements

The first two years of this research were supervised by Prof Ian Sommerville. Ian oversaw the experimental design, methodology development and the first two years of data collection. Ian continued his interest and support of this work after moving to St Andrews University. Ian’s support, guidance and insight gave this research direction.

Dr Simon Lock supervised from the analysis through to final thesis submission. His guidance through this critical period and assistance in getting the work published is greatly appreciated and turned the work from research in progress into a PhD thesis.

I also benefited from the advice of colleagues, input of participants and support of MSc students’ supervisors. Amongst those who provided a sounding board were; Prof Alan Dix, Dr Pete Sawyer, Dr Joe Finney, Dr John Mariani, Kiel Gilleade, Danny Hughes, Saleh Al-Sulmi, Lorna McKnight, Ben Green and Kris Welsh.

The thorough reviewing and recommendations of the examiners, Prof David Budgen and Dr Alessandro Garcia provided the final polish and an enjoyable viva. Where editing inadvertently reduced the information provided on methodology the examiners provided a focus on the future of this work and the need to facilitate replication and not just understanding.

This thesis builds on ideas from my prior work under the supervision of Dr David Squire and Dr Kevin Korb at Monash University, Australia. Without their initial guidance and support this topic could not have been explored. Prof. John Rosenberg encouraged me to undertake a PhD and recommended Ian Sommerville, without the dean’s encouragement the move half way across the world may not have happened.

The challenges of life away from home are many and daunting. The Jewish Society in Lancaster provided a warm welcome, Flora Hoori (then chair) and Prof. Stanley Henig (senior member) gave me support and rapid integration into the community. In the graduate student world Prof. Maurice Kirby (Principle of Graduate College) and Dr James Groves played a similar role. Stanley, Maurice and Rev. Malcolm Weisman provided the sort of concern for my welfare one normally only expects from a parent.
Finally, thank you to my parents, Charmian and Michael, and to my brothers Stuart and Danny. Four years away from home is a long time, and I have missed you all, though from the number of times I called home you might not realise how much.

Andre Oboler, Lancaster, July 2007
Table of Contents

1 INTRODUCTION

1.1 MOTIVATION AND FOUNDATIONS TO THIS WORK

1.1.1 The Software Development Life Cycle for Research

1.1.2 Research Process Optimization Framework

1.1.3 Improvement Based on the Framework

1.1.4 Research Goals

1.2 NATURE, LIMITATION AND APPROACH OF THIS RESEARCH

1.3 NOVEL CONTRIBUTIONS OF THIS WORK

1.4 THESIS STRUCTURE

2 MEASUREMENT, IMPROVEMENT AND PROCESS BACKGROUND

2.1 METHODS OF MEASURING IMPROVEMENT

2.1.1 Mathematical Metrics

2.1.2 Software Metrics

2.1.3 Metrics and measurement in this research

2.2 IMPROVEMENT IN THE RESEARCH ENVIRONMENT

2.2.1 Software Engineering for Research

2.2.2 Legacy properties in the Research Environment

2.2.3 Thoughts on Process Modelling

2.2.4 Measurements of Success

2.3 CORE PROCESS CONCEPTS

2.3.1 The Nature of a Process

2.3.2 Defining Process Improvement

2.3.3 Meta-Concepts: the Process of Process Improvement

3 A REVIEW OF PAST DEVELOPMENTS

3.1 DEVELOPMENT OF THE PROCESS IMPROVEMENT PARADIGM

3.1.1 Taylorism and Scientific Management

3.1.2 The Humanist School and the Hawthorne Effect

3.1.3 Total Quality Management (TQM)

3.2 EMERGENCE OF TECHNOLOGY AS AN ENABLER OF CHANGE

3.2.1 The Rise of Computing: Improvement Through Automation

3.2.2 The Shift to Software Engineering

3.3 CONVERGENCE OF CHANGE AND PROCESS IMPROVEMENT

3.4 IMPROVING THE PROCESS OF PROCESS IMPROVEMENT

3.4.1 Process Programming

3.5 SUMMARY

4 DISCUSSION AND DEVELOPMENT OF KEY THEMES

4.1 BUILDING AN IMPROVEMENT PARADIGM

4.2 USES AND LIMITS OF AUTOMATION

4.3 MAKING SOFTWARE ENGINEERING FIT FOR PURPOSE

4.4 TECHNOLOGY SUPPORTED PROCESS IMPROVEMENT

4.4.1 Organisational Approach

4.4.2 CMM and the CARE Framework

4.4.3 Existing Personal Approaches

4.5 THE PROCESS FRAMEWORK

4.6 PROCESS DESCRIPTORS
8 EVALUATING COMPONENTS OF SERE .............................................................. 160

8.1 THE PROCESS MODEL FOR SUPPORTING RESEARCH .................................. 160
  8.1.1 Developments Since the Final Cycle .......................................................... 160
  8.1.2 Process Model and CARE ........................................................................ 161
  8.1.3 Future work on the Process Model ............................................................ 162

8.2 THE PERSONAL PROCESS MODEL ............................................................... 162
  8.2.1 The Integration Benefit ............................................................................ 165
  8.2.2 The Organisation Benefit ........................................................................ 165
  8.2.3 The Communication Benefit .................................................................... 166

8.3 HIGH LEVEL DIAGRAMS ............................................................................. 166
  8.3.1 Over Engineering Dangers ....................................................................... 167
  8.3.2 Under Engineering Faults ......................................................................... 167
  8.3.3 Reuse and High Level Diagrams ............................................................... 168
  8.3.4 Diagrams and Technical Reviews ............................................................. 168
  8.3.5 Knowledge Gap ....................................................................................... 168

8.4 DOCUMENTATION AND DOXYGEN ............................................................ 169
  8.4.1 Well Documented ..................................................................................... 169
  8.4.2 Under Documented .................................................................................. 169
  8.4.3 Inherited Code .......................................................................................... 170
  8.4.4 Documentation and Technical Reviews .................................................... 171
  8.4.5 Further Feedback on Documentation ...................................................... 172

8.5 THE CODING GUIDELINES .......................................................................... 172
  8.5.1 Ambivalence to Commenting .................................................................... 173
  8.5.2 Adapted Adoption .................................................................................... 174
  8.5.3 Impact of the Guidelines ......................................................................... 176

8.6 TECHNICAL REVIEWS ................................................................................ 176
  8.6.1 Technical Reviews in the Case Studies ...................................................... 178

9 MEASURING SUCCESS .................................................................................... 183

9.1 FRAMEWORK OF SUCCESS ......................................................................... 183

9.2 ADOPTION OF SERE ................................................................................... 186
  9.2.1 A Non-Participant’s View: Case 0625 ....................................................... 187
  9.2.2 An Inactive Participant’s View: Case 0529 ............................................... 189
  9.2.3 View of Participants ............................................................................... 190
  9.2.4 Further Discussion .................................................................................. 193

9.3 FORMAL ASSESSMENT (EFFECT ON GRADES) ............................................ 193
  9.3.1 The Improvement Heuristic ..................................................................... 194
  9.3.2 Preparing the Data .................................................................................. 195
  9.3.3 The Combined Data Set ............................................................................ 195
  9.3.4 The Third Year’s Data ............................................................................. 198
  9.3.5 The Second Year’s Data ......................................................................... 198
  9.3.6 The First Year’s Data ............................................................................... 198

9.4 STUDENTS’ PERCEIVED COSTS AND BENEFITS ..................................... 200
  9.4.1 Case 0405 ............................................................................................... 200
  9.4.2 Case 0628 ............................................................................................... 201
  9.4.3 Case 0629 ............................................................................................... 202
  9.4.4 Case 0522 ............................................................................................... 203
## List of Tables

Table 1 Investigation modes ................................................................. 15
Table 2 Metrics used in this research......................................................... 27
Table 3 ISO90003 summary ................................................................. 47
Table 4 Use of FTRs by Australian Academics (Oboler 2002) ...................... 86
Table 5 Dangers to research without software engineering .......................... 119
Table 6 Projects students thought could be extended 2003-2004 .................. 122
Table 7 Projects students thought could not be extended 2003-2004 ............... 123
Table 8 Final MSc Survey responses from all students 2004-2005 ................. 124
Table 9 Final MSc Survey responses from all students 2005-2006 ................. 126
Table 10 Relative usefulness of process descriptors 2003-2004 .................... 130
Table 11 Student and supervisors views on resetting projects 2003-2004 ....... 131
Table 12 Relative usefulness of process descriptors 2004-2005 ...................... 132
Table 13 Relative usefulness of process descriptors 2005-2006 ................. 136
Table 14 Cases referred to in this thesis ................................................ 146
Table 15 Stakeholders and their goals ...................................................... 184
Table 16 Researcher’s and other stakeholder’s benefits ............................... 185
Table 17 Population of samples over three years ........................................ 187
Table 18 Summary data for 2004-2006 ....................................................... 195
Table 19 Comparison of means (T-test) ...................................................... 197
Table 20 third year delta improvement data .............................................. 198
Table 21 second year delta improvement data ........................................... 198
Table 22 First year delta improvement data .............................................. 199
Table 23 Projects where students felt their work could be extended ............... 209
Table 24 Projects where students felt their work could not be extended .......... 210
Table 25 internal validity threats ............................................................. 215
Table 26 external validity threats ............................................................ 215
Table 27 construct validity threats .......................................................... 216
Table 28 Final MSc Survey responses from the Experimental Group 2003-2004 .... 294
Table 29 Final MSc Survey responses from the Control Group 2003-2004 ....... 295
Table 30 Final MSc Survey responses from the non-participant Group 2003-2004 . 295
Table 31 Usefulness of process descriptors 2003-2004 ............................... 298
Table 32 Average usefulness of process descriptors 2003-2004 ..................... 299
Table 33 Weighted usefulness of process descriptors 2003-2004 .................. 299
Table 34 Weighted average usefulness of process descriptors 2003-2004 ......... 299
Table 35 Weighted average of all process descriptors 2003-2004.................... 300
Table 36 SERE Components ................................................................. 311
Table 37 Cases and benefits ................................................................. 311
Table 38 Cases and project factors ......................................................... 312
Table 39 Summary Data for 04-06 without outliers .................................... 312
List of Figures

Figure 1: The RAISER / RESET SDLC ................................................................. 4
Figure 2: Effort vs. progress pipe model, an example of a hypothetical research process ......................................................... 7
Figure 3: Effort vs. progress bar graph, an example of a hypothetical research process ..................................................................................... 8
Figure 4: The Five level of Software Process Maturity ................................ 49
Figure 5: The four levels to the PSP ............................................................... 51
Figure 6: The research process ........................................................................ 67
Figure 7: The research process as an instantiation of a Process Descriptor ........... 68
Figure 8: The process Improvement Process ................................................ 69
Figure 9: Process of Improving Process Improvement Processes ................. 70
Figure 10 Components of SERE .................................................................... 81
Figure 11 The final process model ................................................................. 82
Figure 12: The first year process ..................................................................... 96
Figure 13: The second year process ................................................................. 103
Figure 14: Process Improvement Process ....................................................... 108
Figure 15: The third year process ................................................................. 109
Figure 16 The Personal Process Model: An example ..................................... 164
Figure 17 Box plot of 2004-2006 by group .................................................... 197
Figure 18 Box Plot of 2003-2004 ................................................................ 200
Figure 19 Personal Process Model: The Research Process for this work .... 284
Figure 20 The Process Model: The Components of SERE ......................... 285
Figure 21 Box Plot 2004-2006 without outliers ............................................. 313
**List of Abbreviation**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARE</td>
<td>Capability of Academic Research Environment</td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning systems</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>FTR</td>
<td>Formal Technical Review</td>
</tr>
<tr>
<td>OO</td>
<td>Object oriented</td>
</tr>
<tr>
<td>PIPSI</td>
<td>Process for Improving Programming Skills in Industry</td>
</tr>
<tr>
<td>PPM</td>
<td>Personal Process Model</td>
</tr>
<tr>
<td>PSP</td>
<td>Personal Software Process</td>
</tr>
<tr>
<td>QAA</td>
<td>Quality Assurance Agency</td>
</tr>
<tr>
<td>RAE</td>
<td>Research Assessment Exercise</td>
</tr>
<tr>
<td>RAISER</td>
<td>Reactive Assisted Information Science Enabled Research – the research part of the RAISER / RESET SDLC</td>
</tr>
<tr>
<td>RESET</td>
<td>Research Enabled Software Engineering Techniques – the stabilisation phase of the RAISER / RESET SDLC, where re-engineering occurs</td>
</tr>
<tr>
<td>SDL</td>
<td>Software Development Laboratory – the group responsible for implementing RESET.</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Life Cycle</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>SERE</td>
<td>Software Engineering for the Research Environment</td>
</tr>
<tr>
<td>TQM</td>
<td>Total Quality Management</td>
</tr>
</tbody>
</table>
1 Introduction

Software built for research (e.g. to prove a concept, test an algorithm, experiment with innovation etc) is often built in an unsystematic manner. As new people take over or extend existing projects, knowledge is lost. With increased time pressure placed on research students it is becoming harder to do significant research. While the lifetime and reuse of software in other fields is extended, research software appears to have inherent legacy like properties holding it back. In other software development environments software engineering emerged as the solution to these problems. The nature of conventional software engineering however, does not meet the needs of research and the constraints of the university research environment. As a result the progress of science in many areas of computer science is retarded.

This research examines ways of improving the computer science research process in the short term (as research is conducted) while also adding value that can help future researchers. Specifically we aim to improve the research process through the identification and introduction of appropriate software engineering techniques and a process improvement paradigm specific to the needs of computer science academic researchers. To be adopted, techniques for the academic environment need to give maximum benefit to researchers while requiring little effort from them, specifically techniques that reduce wasted time while providing additional benefits to the researcher are needed. A feature of the academic environment is that the end goal and implementation are constantly changing, and as a result the traditional approaches of software engineering usually fail to be adopted. Researchers, particularly when under time pressure, are very cautious in adopting new techniques.

For a technique to be adopted its benefit must be clear and its cost in effort low. In this work tools and techniques that meet this requirement are identified, developed and tested to see which researchers find useful and are willing to use. The dOxygen software in particular has been used as a means of facilitating the adoption of coding standard, process modelling, and technical reviews. We have examined the needs of researchers and the acceptability of tools and techniques through the use of extensive case studies. Our case study subjects are MSc students undertaking five month
research projects. The researchers themselves accept or reject changes depending on their needs and those of their environment.

The concept of a tool (or more correctly a process descriptor) in this research includes guidelines, templates and approach recommendations. Adoption is taken to mean the use of tools presented, as well as the adoption of similar tools and approaches that researchers are inspired to find, develop or adapt from those presented to them. As more resources were made available some researchers engaged at a deeper level and their requests became more focussed and specific. The focus on techniques with a low cost was reinforced. Feedback and results have indicated that a tool or technique must have a low barrier to benefit identification, a low barrier to adoption, and a low barrier to use. As with improvements in the teaching process, to improve the research process significantly, a researcher must become critically aware of what they intend doing and how it might be done better. We show this critically reflective role in our Process Model for Supporting Research and facilitate it through the development of the Personal Process Model (pg 162), an approach which focussing researchers attention on the process view of their work.

The creation of the Personal Process Model as a tool for researchers is one of the most significant contributions of this work. It incorporates the tools, techniques and research tasks in a meaningful and systematic way that is unique to each researcher. It is a high level approach that introduces software engineering to the research process through abstraction, design and reuse. We believe the personal software process combined with the other tools presented in this work, or with tools yet to be created, is a significant and novel approach to improving the research process and deserves further investigation beyond the scope of this thesis. We discuss some ideas for its future development in our future work section.

In the remainder of this introduction, we present the motivation for this work, introduce some of the underlying ideas and discuss the nature and limitations of our approach. We also present some of the novel contributions of this work and introduce the structure of this thesis.
1.1 Motivation and Foundations to this work

Improvements in the way research products in the computing discipline are created and maintained may allow better results, faster advances and more efficient research than the present norm. Improvements may allow more researchers to work beyond current limitations created by their environment and time constraints and allow greater collaboration and greater continuity and reuse of research.

In a past research project we examined the use of software engineering in computer science research, the USE CSR project (Oboler 2002). This was achieved through surveys of academics in Australia and the United States, a case study examination of three research projects and interviews with various academics and software engineering experts. This work painted a picture that showed a broad lack of software engineering in academic research. We discuss this past work in more detail in section 2.2.1 (pg 29), but introduce in this introduction our Software Development Life Cycle for research. This SDLC provides one of the foundations underpinning our current work.

Another foundational concept is the Research Optimization Framework. This model of the research process is a philosophical construct that allows us to separate those aspects we can improve from those aspects of research that should be left alone. It is introduced in Section 1.1.2, and in Section 1.1.3 we discuss approaches to improvement based on this framework. Both these foundations are new to this research, but are introduced here to set the context of our work.

This thesis seeks to build on our previous work, and through a larger selection of case studies, gather a more in-depth understanding of the research environment and why researchers choose to adopt or not to adopt new approaches and tools. As previously mentioned, we also introduce our own tools and approach and evaluate their acceptance, use and benefits. While undertaking this work we’ve aimed to continually expand our understanding of the nature of the research environment, its problems and needs, and continually improved our approaches and tools to make them more compatible with the research environment and better tailored to the goals and constraints of researchers.
1.1.1 The Software Development Life Cycle for Research

Also resulting from our earlier work was the creation of a new Software Development Life Cycle (SDLC) for the research environment. The SDLC, known as the RAISER / RESET, or “pulley” SDLC, was designed to achieve a separation between research effort and stabilisation effort. It was based on the results collected in the USE CSR project and also recommended some principles for the introduction of appropriate software engineering approaches that would meet the needs of the research environment. The approach we have used in our present research is based on and greatly enhances the SDLC from our previous work. The division between research and stabilisation can be seen in Figure 1. The top half of the diagram is the research intensive RAISER part, standing for Reactive Assisted Information Science Enabled Research and the bottom part is the reengineering or RESET part (Research Enabled Software Engineering Techniques). The present research specifically focuses on RAISER and the inclusion of software engineering in initial research (as shown in Figure 1) to facilitate RESET work and aid follow-on research.

![Figure 1: The RAISER / RESET SDLC](image)

As explained in our past work (Oboler 2002; Oboler, Squire et al. 2004), a RAISER approach should be a set of recommended tools, processes and practices from (or adapted from) traditional software engineering and thought to be compatible and of
value to small team research projects (e.g. three or fewer researchers working on a project). As the name suggests, the methods are chosen to be adaptable and reactive to changes in the research rather than trying to constrain the research as traditional software engineering does. They should also be designed to assist the researcher, i.e. be of benefit now, not just for re-engineering. The current project further developed these ideas and tried implementing a number of tools thought to fit this description in order to test their adoption. In the current work, we also adopted a process-based view of research and began to aim not just for improvements to the product, but also improvements to the research process. Perhaps the most significant development in the present work is the concept of Personal Process Modelling that made researchers responsible for mapping out their own research process and seeking ways to improve it. Another development in the current work is the Process Model for Supporting Research, a systems approach to implementing the RAISER / RESET SDLC concept in an organisation setting. Both these ideas build on a foundation based on the separation between research and development, as introduced by the RAISER RESET SDLC.

1.1.2 Research Process Optimization Framework

Another foundational idea to this research is the flexibility of research goals. It is not the distance from a particular goal that is important in research; rather it is the progress itself that is significant. Although almost a cliché today, a small quote from Alice in Wonderland sums the situation of arbitrary endpoints perfectly. While asking the Cheshire Cat for directions the following conversation ensues:

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don't much care where--" said Alice.
"Then it doesn't matter which way you go," said the Cat
"--so long as I get SOMEWHERE," Alice added as an explanation.
"Oh, you're sure to do that," said the Cat, "if you only walk long enough."
- Alice in Wonderland

The conversation between Alice and the Cheshire Cat can be regarded as a parable for the research process which starts out with a direction but quickly diverts down more
interesting tangents. This does not mean the researcher is lost; the concept of being
lost only applies when a particular destination is in mind. Indeed the wandering from
the initially planned path may in fact be the optimal way to make a significant
contribution to science. The issue of shifting goals needs to be seriously engaged with
when putting together a framework to discuss optimization and improvement of the
research process.

To improve research we need some concept of what research itself is, and some way
of distinguishing between better and worse approaches. Our investigation uses
research skills ranging from empirical software engineering, through to those more
commonly associated with the social sciences (such as case studies, interviews and
surveys). What we present here, however, is a philosophical look at improving
research, what this means, and abstractly, how one can go about doing it.

To facilitate our discussion we introduce a metric that is difficult to measure
generally, and in fact impossible within the constraints of our research that takes place
in-vivo. The measure is a degree of effort, specifically the effectiveness of this effort –
how much of it is forward momentum, and how much is “wasted” on non-creative
tasks. To return to our Alice example, researchers are concerned with getting
somewhere. For many, the exact nature of where is not important, so long as it IS
somewhere i.e. it meets the criteria of an original contribution. This can surely be
done provided one keeps working for long enough.

For both students and universities there is increasing pressure to place bounds on the
time allowed to complete a PhD. For researchers who have their PhD, their
publication record is a key to promotion and an indicator of the health and prestige of
this department. For both students and academics there is pressure to get results faster.
The option of working as long as it takes is rapidly becoming a quirk of the university
system to be relegated to history. Any discussion of process improvement therefore
needs to examine the rate of improvement, or new knowledge production compared to
time.

Our Pipe Model (See Figure 2), created as part of this research, shows the rate of
knowledge acquisition compared to effort. The Pipe Model shows the research process
as a channel from the start (A) to the successful completion of a unit of research (B). The line’s width is variable and measures effort spent to move a fixed distance closer to the goal. The length of the line is an indication of progress in terms of new (to the researcher) knowledge gained through the research process. At some point information that is new to the research will be new to the research community as well. When a research is enough steps ahead of the rest of the community (or runs out of time) the researcher may decide to publish and end this particular research task.

Figure 2: Effort vs. progress pipe model, an example of a hypothetical research process

The X-axis represents Standard Knowledge Units (SKUs). The Y-axis is effort per SKU of progress (standard effort units or SEU)

An alternative view to the channel of varying widths (Figure 2) is a graph combining forward momentum with overhead, like the pipe this varies with each stage (Figure 3). With a constant force forward, progress through the pipe model would be determined by the width, and speed of knowledge acquisition would vary between sections. The example shown (in both diagrams) can be explained as: At the start (a) is a gradual increase in the amount of effort required to make progress as the researcher moves from basic easy-to-find knowledge to harder-to-find knowledge. There follows a period (b) of experimentation during which large effort is expended to create new knowledge. The new knowledge is placed in the context of existing knowledge and deeper understanding is gained (c). There may be a hurdle (d), a period of time when the knowledge doesn’t agree and extra effort is expended to overcome this before returning to steady progress (e) which tails off as all the bits fit into place, or avenues of new knowledge are excluded from the present investigation. Towards the end (f) a large amount of effort is required to reach the goal, with very little new knowledge gained (this is the writing up phase).
Figure 3: Effort vs. progress bar graph, an example of a hypothetical research process

The top part of each bar represents the progress forward (in terms of SKUs) the total high of the bar represents the total effort expended in that stage. The lower part of each bar is the Standard Effort Units or SUE required to generate the SKU at that stage of the process. This is the same fictional example as used in the pipe model above.

There is no way of measuring the standard knowledge units used in the Research Process Optimization Framework, but the hypothetical model allows a degree of abstract thinking on how such a unit could be optimised. The aim of an improved process is to reduce the amount of Research Work need to move from start to finish, i.e. to reduce the surface area of the models in Figure 2 and Figure 3. With progression from start to finish measured in standard knowledge units we could compare absolute knowledge gained and the rate of progress between individuals in gaining new knowledge.

We cannot create a true measure of a SKU as this is a measure of human understanding and cannot be directly accessed. Even if we were to monitor a change in brain activity, there would be no way of knowing how much of that activity was
heading in the right direction, towards the end goal. In fact what may for a time look like motion in the right direction, could in the end be revealed as a diversion.

The closest we can get to these measurements (in a discrete time period) is a person’s own measure of their study effort applied and the amount of relevant knowledge they gained. Of course the relevance must be measured with hindsight. If the persons measure is considered to be fairly consistent over time, it is then possible to measure both the knowledge displacement in the researcher and the Research Work done. These can then be compared to other research projects of the same researcher. The overhead in such measurement is however high and this concept is not examined further in this research.

As must now be apparent, the model presented in this section (Figure 2 and Figure 3) is a hypothetical example not meant as a representative pattern, but only to highlight the concept of constant effort producing varying degrees of forward progress, depending on the overhead effort. With various defined stages having variable overheads, the model does however highlight that various stages and overheads could be targeted for effort reduction. The effort that is being expended on overheads needlessly or inefficiently is what we refer to as “wasted effort”. One of the goals of this work is to reduce such wasted effort; the three ways to do this are discussed in the next section.

1.1.3 Improvement Based on the Framework
Wasted effort can be reduced in three ways; move the starting position further along (give researchers a head start, e.g. through improved reuse), bring back the ending position (e.g. by recognising and allowing a researcher to take the first exit), and by reducing the effort factor during the research (e.g. through automation). We now discuss these three options in more depth, and how they apply to this work.

Moving the start position
Removing the need of some prior knowledge allows a researcher to enter at a position closer to the end goal. The model reflects this perfectly, the researcher is closer to the goal, but does not gain as much knowledge over-all.
The use of abstraction to limit the amount of information (knowledge) required is a key factor in software engineering, particularly in OO. The danger is when the abstraction “leaks” and the prior knowledge must then be learnt in order to patch it (Spolsky 2002). In order to limit leakage, we seek to provide abstractions to the researcher that are modular, have good cohesion and loose coupling. This applies equally to software and concepts.

An analogy: In the example of a tailor, existing clothes can be used as a starting point in creating a “new” item of clothing. In an extreme case the tailor may simply shorten the legs, or change the buttons. In other cases the tailor may create a new type of garment out of the material of the old one. In both cases this is a cheaper solution. It may not be as perfect as a product made from new material, but it will almost always be cheaper (if the provided garments are of good enough quality).

Our approaches to documentation in particular aim to increase the reusability of past research products. Other approaches like Technical Reviews aim to help improve design, increasing adaptability in the longer term.

Moving the end position
As the end position is often an unknown and in many respects unimportant, the key is to recognise when an end point has been reached.

Continuing our analogy: Our tailor has a dress and is decorating it: – He starts by adding sequins around the neck in a thick band, intending to continue adding them right down the length of the dress. Time runs out (the customer arrives), the customer looks at the dress and is satisfied, the tailor ends his work. Note that if the start position had also been moved the tailor’s entire job may have been 30 minutes worth of affixing sequins. The customer will be aware of this and the resulting value of the tailor’s contribution to the work is far lower and may even be considered trivial. There is an obvious danger in only doing trivial work. This can be avoided by moving the end point further away, i.e. being more ambitious with one’s research.

The use of a Personal Process Model and the recording of rationale through the coding guidelines allow a researcher to discuss not only their product, but their contribution
to research more effectively. While the decision of where an end-point is can only be made by the researcher (and on advise from other researchers), a more complete understanding of the work completed and planned can aid such decision making.

Reducing the effort factor
In his famous paper “No Silver Bullet” Fred Brooks says that “the complexity of software is an essential property, not an accidental one” (Brooks 1986). He goes on to explain this as meaning that software is pure “thought stuff” and to abstract away its complexity of concept is often the same as abstracting or wishing away the software itself. We cannot reduce complexity of the concept, or the effort required for a researcher to come to grips with this concept.

However, the most valuable part of the programming or more generally the research (i.e. the creation of understanding and eventually new ideas) is often not the part that takes the most time. “With any creative activity comes dreary hours of tedious, painstaking labour, and programming is no exception” Brooks acknowledges (Brooks 1995). The effort spent on these accidental parts of the process can and should be reduced.

We are not asking people to work harder but rather trying to change the vector of work so more of it can be expended along the path between start and end, i.e. so more of the effort is spend gaining new knowledge. We are asking people to work smarter.

We return to our analogy of the tailor: - if a tailor is skilled his process for making a well-fitted garment will be efficient – yet unique to each garment. The skilled tailor has a process for making his clothes and this uses tools and often repeated sub-processes. The task of improving the research process in computer science academia is the task of equipping a group of tailors with the appropriate tools and reusable procedures, and aiding them in constructing their unique approach to the problem they are working on.

Tools and techniques to minimise the non-essential effort exist, but have often not been tested for impact in the academic environment. Our past work shows they are not in common use, and this new work has incorporated some of them into our
experimental subjects research projects and recorded the results. The bulk of our observations are on the effectiveness of such approaches and tools, either existing ones we have adapted to the research environment, or new ones we have created during this research.

1.1.4 Research Goals
In this thesis we present and evaluate a new process-based approach to software engineering designed to meet the needs of academic Computer Science researchers. The core objective is to examine whether our software engineering approach is an improvement, and how it affects the research process, the research product and the researchers themselves. We examine issues such as adoption, costs and benefits from the researchers’ point of view, and benefits to the quality of the research product, including aspects such as reusability and readability. From a systems perspective we examined some of the issues related to implementation on an organisational basis. We aim to explore process improvement through our SERE approach from the perspective of the researchers as gathered in surveys and case studies (presented in Chapter 7), the new approaches (presented in Chapter 5 and evaluated in Chapter 8), and the success of SERE as a whole (presented in Chapter 9).

In this work we will:

- Examine the suitability of our overall approach to the research environment
- Develop and adapt tools and approaches to meet the needs of the research environment
- Evaluate and improve the components in an evolutionary manner
- Assess the costs and benefits of the SERE approach to process improvement

Benefits we will examine as aspects of an improved process included:

- Knowledge retention within and between projects
- The use of systematic approaches
- Reduction of avoidable problems
- Reusability and adaptability
- Improved efficiency of the process and reduction in wasted time
- Researchers perception of costs and benefits
Chapter 1

Introduction

- Increased quality of the research product as measured by markers

We believe a combination of process improvement and software engineering in the Research Environment will enable more research to be conducted and larger goals to be achieved. It will increase the useful lifetime of some research prototypes and increase their potential for reuse, thereby creating new synergies between projects. This in turn will mean greater value for money invested in computer science research as more effort is spent on the essential rather than the accidental problems. We believe a conscious focus on the research process and its improvement will lead to better research. This work aims to make a contribution to this effort in understanding research and exploring ways of improving the research process.

1.2 Nature, limitation and approach of this research

In this research MSc students undertaking five month research projects were used as experimental subjects. The sample was self selecting, but about half of each cohort of students volunteered. The large number of case studies ensured replication of results. The number of test subjects also allows empirical results to be collected in addition to more focused case study evaluation. Surveys and interviews with PhD students and some lecturers add additional expert opinion. Our research uses a mixed methods approach, using both qualitative and quantitative results to measure the change and its impact on the outcomes. The use of various subjects and methods of data collection allows a deeper and more complete understanding of the subject through triangulation (Patton 1990).

Case studies are defined by Yin as “an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin 1994). Case studies are therefore the most appropriate approach to investigate and understand the research process as it occurs. An analysis of research across three computing disciplines (Glass, Ramesh et al. 2004) shows that though it is seldom used in computer science research (0.2%), the case study approach is slightly more common in software engineering research (2.2%) and is a popular technique in the area of Information Systems (12.5%).
Data for case study analysis is collected in an all encompassing manner and in our work we focused on the use of observations, structured interviews (early in each cycle), surveys, semi-structured interviews (the pre-planned questions often tailored to the student based on survey responses they provided), technical reviews, and inspection of artefacts (i.e. diagrams, dissertations, code, and students’ websites).

Our research also seeks to apply a treatment (the SERE approach) to some students’ research process. The impact of the new approach is best examined as an experiment, though for reasons explored later in this thesis, a quasi-experiment was considered more practical and more ethical. The ability to interpret the experimental data in light of the case study data adds depth to this research. Field experiments (such as this) were not found in the computer science literature sample in the research by Glass, Ramesh and Vessey (2004) and were seldom found in software engineering research (less than 1%) or Information systems research (1.6%). This highlights a deficiency in the field (across all three disciplines), but one which this research aims to avoid.

In Table 1 we present the main approaches used in this research. Where surveys have been used, the surveys themselves are presented in the appendix. Where case studies have been used, the case study design is provided in the methodology section of this thesis. Case study and experimental designs were created and reviewed prior to practical work (including data collection) commencing. Additionally a number of case study plans were created but not implemented as the scope was refined.

<table>
<thead>
<tr>
<th>Under examination</th>
<th>Mode</th>
<th>Unit of analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing ideas on process improvement</td>
<td>Literature / history survey</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>(Exploratory research)</td>
<td></td>
</tr>
<tr>
<td>Research experience</td>
<td>Case study</td>
<td>Single researcher</td>
</tr>
<tr>
<td></td>
<td>(Exploratory research)</td>
<td></td>
</tr>
<tr>
<td>Adoption and diffusion of SERE components</td>
<td>Case study (largely based on two survey)</td>
<td>A cohort of MSc students</td>
</tr>
<tr>
<td></td>
<td>(Descriptive research)</td>
<td></td>
</tr>
<tr>
<td>Reusability (1st year only)</td>
<td>Case Study</td>
<td>A single project</td>
</tr>
<tr>
<td></td>
<td>(Explanatory research)</td>
<td></td>
</tr>
</tbody>
</table>
Table 1 Investigation modes

<table>
<thead>
<tr>
<th>Perception of Software Engineering</th>
<th>Survey (Exploratory research)</th>
<th>Two cohorts of MSc students (all students)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perception of SERE</td>
<td>Case study (Exploratory research)</td>
<td>Three cohorts of MSc students (all students)</td>
</tr>
<tr>
<td>Impact of SERE on research</td>
<td>Experiment (year 1) (Explanatory research)</td>
<td>A group of students (participants compared with non-participants)</td>
</tr>
<tr>
<td>Impact of SERE on research</td>
<td>Quasi-experiment (years 2 &amp; 3) (Explanatory research)</td>
<td>A group of students (participants compared with non-participants)</td>
</tr>
<tr>
<td>Effect of individual process descriptors</td>
<td>Case study (Descriptive research)</td>
<td>A process descriptor</td>
</tr>
</tbody>
</table>

Case study data from a number of different cases may at times have been collected simultaneously, particularly through the use of structures interviews where the interviews were often broken up into a series of themes. In some cases data of a different for was collected explicitly for one case study, for example the use and interviewing of observers in technical reviews. These focussed not on the project being reviewed, but on the process of the review itself. In a similar fashion, data effecting the experiment and quasi-experiment were collected through the case studies. Particular attention was paid to confounding factors that may have prevented adoption or participation. These are discussed later in the thesis.

The case studies were strongly coupled with the schedules of the subjects under investigation, and the research schedule for this work was designed to fit within the constraints of the experiment. Experimentation and data collection had to occur when the opportunity presented itself. Experiments started when the subjects were ready, not when the experimental design or tools were ready. For this reason the usefulness of our approach was lowest in the first year and highest in the third year. A comparison can be made to lectures delivered by the same person and continually revised and improved each year. One side effect (discussed in more depth in the thesis) is that not all the results from the first year of our experiment are comparable to
the later two years, due in part to change in experimental design and the relative immaturity of the approach at that point.

In order to fully engage with this research is has been necessary to think about the research process as an object. This, however, can lead to confusion when the word “process” is being used both as the noun and as a verb at various times. There are also various different processes at play simultaneously. At the lowest level there is an MSc student going through the process of doing their research. At the top level there is the process of improving our experimental methods each year.

To avoid confusion, it is important to keep clear:

- The subjects of the experiments (i.e. an MSc student doing an MSc project)
- The methodology under investigation (i.e. process improvement in a RAISER frame work that emphasises “maximum immediate benefit, with minimum cost”)
- The methodology of the investigation (i.e. how this research is done – the process of improving process improvement)

We believe the choice of Masters students as research subjects was necessary in order to put a time constraint on data collection and allow multiple cycles of the experiment to run. This has been of immense benefit to the development of our approach. While the motivation of Masters students may differ from that of other researchers, we believe the data collected shows the SERE approach can improve the research process and product. The types of advantages gained by MSc students undertaking their research projects would be of benefit to researchers more generally. The SERE approach is designed for general applicability in an academic research environment and many of the advantages (such as knowledge retention, support for collaboration, etc) have been designed to give even greater benefit on longer projects such as PhDs work, or longer term departmental projects. Future work would need to access the success of more general applicability now that the approach has been created and tested on one class of researcher (MSc Students).

The major difference between MSc students and other researchers is the relatively short nature of the research process. We believe the benefits demonstrated in this
thesis may indeed be greater than we’ve reported if PhD students and academic staff adopted SERE. Where it has been feasible we have used the approaches we present in our own research, and while this is by no means an objective opinion, we believe them to have been helpful here as well.

Although focusing on the academic research environment, the tools and approaches may be applicable in other settings as well. The needs of some industrial research environments, particularly those run on the campus model, may be addressed through the SERE approach. There may also be lessons from this work for software engineering and computer science education. The SERE approach could, for example, form a component in, and give structure to, a research methods subject at undergraduate or masters level. These additional benefits are, however, beyond the scope of this thesis.

1.3 Novel contributions of this work
This research contributes to our understanding of the nature of computer science research, validating and significantly expanding ideas in our past work (Oboler 2002).

SERE, a software engineering approach for academic research based on the RASIER / RESET SDLC, was created, tested and improved in an evolutionary manner. This approach has shown both significant acceptance and demonstrable benefits for researchers. This approach demonstrates that software engineering can be beneficially applied in the academic research environment.

A meta process model, the Process Model for Supporting Research, was created to enable departments to support the research processes of their academics. This model was tested on a reduced scale and shown to be practical. This research was published as (Oboler, Sommerville et al. 2006).

The concepts of personal process models for academic researchers were created, and ways of integrating this with the software development environment are presented in this thesis. Results from a number of case studies where the approach was used are examined.
The use of an adapted type of formal technical review in the academic environment was examined. This was presented at the CAQDAS 07 Conference (Oboler 2007).

A documentation standard specific to computer science research was proposed. This was used by researchers over the full three years of the study and published as (Oboler and Sommerville 2007).

Meta level work included an examination of the different types of knowledge in the research environment, and an examination of how improvements could be targeted to maximize benefits and minimize impact. This will be published as (Oboler, Lock et al. 2007) in September 2007.

1.4 Thesis Structure
The structure of this thesis is outlined below. It should be noted that the SERE approach, as presented in Chapter 5, emerged over the course of this research (and as described in Chapter 6 and based on background, analysis and ideas presented in the first four chapters). Chapter 7 discusses the views and opinions of students gathered from surveys and case studies, while Chapter 8 returns to a thematic discussion that evaluates some of the components of SERE as introduced in Chapter 5. Chapter 9 evaluates the success of our approach from a number of different angles before we conclude with Chapter 10.

The full thesis structure which reflects not only the outcomes but the development of the key ideas and the motivation for various improvements is now presented.

Main Thesis
Chapter 1 provides an introduction to the thesis and the conceptual framework in which it takes place. A number of novel contributions are already introduced here as they provide the foundation later work builds upon.

Chapter 2 provides a background on measurement and the software process.
Chapter 3 returns to first principles and draws out lessons from the development of the process approach industry, and the emergency of technology as a force for process change. The concept of process programming is introduced.

Chapter 4 provides a discussion and development on the key themes and lessons from chapters 2 and 3. It is with this background that we seek to bridge the gap between software engineering as traditionally practiced, and the unique requirements of the academic Research Environment.

Chapter 5 provides an overview of the SERE approach, as it exists at the end of this study. It also specifically discusses those components whose effect will be analysed later in chapter 8.

Chapter 6 details our experimental design choices and methodology. The chapter chronologically presents the development of our experiment, the maturing of our understanding and approach, and our reflections on each year of the experiment.

Chapter 7 presents our experimental results. These include surveys conducted with all students before and after the projects, as well as survey data from participants that reflects on the SERE participation experience and the relative benefits of the various interventions used. Also provided is an overview of the case study data as well as a selection of cases provided in more depth.

Chapter 8 provides a discussion on some of the components of SERE. In this section a selection of the process descriptors are introduced and feedback on them from the case studies is presented. We discuss the costs, benefits and students’ impressions of the constituent parts of SERE.

Chapter 9 is an evaluation of the success of the SERE approach as a whole, according to various metric ranging from the effect on students’ marks through to issues of adoption and benefits such as reuse and knowledge retention.

Chapter 10 draws conclusions, discusses the useful contributions made in this work and presents ideas for future work in this area.
Appendices
Chapter 11 presents appendix 1, a collection of the surveys used in this work.

Chapter 12 presents appendix 2, a collection of the process descriptors used and referred to in our experiment.

Chapter 13 presents additional results not included in the body of the thesis. This data is provided for completeness and to separate tangential issues from the main body of the thesis.
2 Measurement, Improvement and Process Background

In this section we discuss the concept of measurement, and specifically how it can be applied to improvement, and what we mean by ‘process’ and ‘process improvement’. We look in detail at the development of the discipline of process improvement from two perspectives: the history of examining process (beginning with “scientific management”) and the history of using machines to improve a task. These two histories converged as computer based systems became an aid to change, and specifically to process improvement. From this section we learn about various methods of improvement and differing approaches. The section review places this in the context of process improvement for the academic research environment and draws out the lessons we can learn from this section of the literature.

2.1 Methods of Measuring Improvement

Having introduced our foundations and goals, the concept of measurement in general, and as it effects this research in particular, deserves further explanation. In this section we discuss the nature of metrics and how they have been used both in software and in academia. The foundations of measurement in both the software field and in terms of measuring academia are weak. The attempt at measurement in both fields could indeed still be considered more art than science. Lord Kelvin (1883) once said:

"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science"

In some respects this thesis represents the beginning of knowledge, as explained in our research goals, it is preliminary work at trying and improving new methods. This is, however, done as empirical research, with a concern towards measurement of the results and impact through a mixed-methods approach using the strengths of both quantitative and qualitative methods. Despite our inclusion of some research methods more commonly found in the social sciences, the specific concern of scientific
measurement of software engineering processes and products is a task of empirical software engineering (Jeffery 2002).

Roberts suggests that, “the major difference between a ‘well developed’ science such as physics and some of the less ‘well-developed’ sciences such as psychology or sociology is the degree to which things are measured” (Roberts 1979). Basili contends that like physics, software engineering is a “laboratory science” (Basili 1996). Both those views ignore the fact that physical sciences work within the rules of nature, where physical measurements are conceptually at least relatively straightforward, while computing and fields like sociology do not have this restriction. In both software development and sociology there is a strong human element, and people often form the largest variable. This makes measurement both conceptually and often practically hard. Despite this, the scientific method from the “well developed” sciences can still be used – provided the human element is taken into account.

To ensure scientific rigour, a formal experiment is needed: a hypothesis must be formulated, data collected, empirical analysis must be conducted, and finally the experiment should be replicated. In the software engineering and computer science fields in general this is often done poorly if at all (Fenton, Pfleeger et al. 1994; Glass 1994; Jeffery 2002) and according to some it is not only unacceptable, but alarming when compared to other fields e.g. the natural sciences (Lukowicz, Heinz et al. 1995). To avoid the common weaknesses this research began with a first principles approach based on recommendations from the literature.

The first step to “measuring things” is to specify what it is we are measuring. The metric must meet certain criteria to qualify as scientifically valid. Numbers are not in an of themselves useful as “the purpose of computing is insight, not numbers” (Hamming 1986). The metric must therefore be converted back into useful information. In short, it must be not only valid but also useful and cost effective to collect (Johnson 2001). In the last few years there has been an increase in the understanding of the impotence of software metrics; however understanding of software measurement is still very limited (Dekkers 2003).
In their paper “Science and substance: A challenge to software engineers”, Fenton, Pfleeger, and Glass (1994), give guidelines for scientific software engineering experimentation:

- Test hypotheses using an experimental design that is appropriate
- Avoid toy problems and toy situations, one cannot necessarily extrapolate
- Ensure there is sufficient data collected and that it is analysed properly. Ensure that the right attribute is measured and on the right scale.

The SERE project was designed to follow these guidelines and meet these requirements. While many of our results are qualitative, the quantity of data and repetition of results as well as the basis of the experimental design makes some of these results quantifiable. It also adds a degree of confidence far larger than that of any one case study or of a simple exploration.

In the next few sections we discuss the idea of metrics and measurement from various angles. We begin with the mathematical definition of a metric, then discuss how the software engineering definition relaxed this definition. Finally we discuss the metrics used in the academic environment and how this thesis should be considered in relation to these approaches.

2.1.1 Mathematical Metrics

In mathematical terms, a metric is a distance function. A geometric space for which there is such a distance function is called a metric space. In order to make meaningful comparisons between objects, they must exist in the same space and their closeness (distance apart) measured using the metric of that space.

In order for a function to qualify as a mathematical metric, it must have 3 properties:

1. Positivity:
   \[ d(x,y) > 0 \] unless \( x = y \) in which case \( d(x,x) = 0 \)

   That is the distance between them must be greater than 0 unless they are (for this metric) the same.

   1 http://www.math.okstate.edu/mathdept/dynamics/lecnotes/node33.html
Symmetry:
\[ d(x, y) = d(y, x) \]
The distance from one point to the other is the same as the distance back

Triangular Inequality:
For all, \( x, y, z \in X \)
\[ d(x, y) \leq d(x, z) + d(z, y) \]
The distance along one side of a triangle is less than or equal to the total distance along both other sides. i.e. a direct path along a straight line between two points is always the shortest path.

2.1.2 Software Metrics
In the software metrics literature the requirements of a metric are often relaxed compared to the mathematical requirements. A software metric is described by Zuse as “a mapping of empirical objects to numerical objects by a homomorphism” (Zuse 1995), that is a mapping preserving all relations and structures. There are many such metrics: by 1995 over a thousand measurements had been proposed (Zuse 1995). Despite this abundance of metrics, there is a “lack of maturity in software measurement” in addition to a lack of standard and useful units of measurements (Zuse 1995). The very concept of metrics is not well understood by either researchers or industry (Zuse 1995; Dekkers 2003).

Some common product metrics include: Number of defects, Function points, Number of objects and lines of code. Process metrics usually match time to a product metric.

Software metrics are generally used as a basis for estimations, a measure of complexity, a measure of quality, or a measure of best practice. These generally relate to progress towards the goal of a finished product, or the quality of the product during the journey.

The most well known use of software metrics for estimation is COCOMO (COstructive COst MOdel) (Boehm 1981). COMOCO, published in 1981, was an approach for estimating the cost, effort, and schedule of a project based on 15 metrics. Additional metrics could be used to account for software reuse. COCOMO and later its replacement COCOMO II were the most widely used approaches to software cost estimation. Even so, the 1998 version of COCOMOII was only “accurate” (within
30% of the actual cost) 75% of the time, and this was an improvement on the 1997 version which was only accurate (within 30%) 52% of the time (Chulani, Boehm et al. 1999).

One major difference between industrial projects for which approaches like COCOMO have been used and academic projects is size. Academic projects are typically far smaller rendering most of the metrics used irrelevant. A less obvious difference was mentioned when we discussed improvements based on the research optimization framework, namely that, in a research project there is often no specific end-product in mind. Even when the overall end-point is clear, there is a lack of precise blueprints and knowledge of how to get there. In terms of quality, to meet the goals of the researcher, their product often doesn’t need to be particularly stable, easy to use or maintainable. These may come later, but for the researcher, focusing on these aspects detracts from their overall productivity (on the scale they are themselves measured on, i.e. research output like new publications). For the research process itself (as it relates to the researcher) we need new metrics that are different and more relevant.

2.1.3 Metrics and measurement in this research
According to Perry, Porter and Votta (2000), “defining and executing studies that change how software development is done - is the greatest challenge facing empirical researchers” in Software Engineering. This is exactly the challenge this thesis attempts to undertake, albeit in the limited field of computer science academia.

The difficulties with this research include the fact that many studies have poor statistical design and don’t scale (Fenton, Pfleeger et al. 1994), the research is conducted over too limited a time period (Fenton, Pfleeger et al. 1994) and that the variety of projects present such differences that comparison becomes difficult (Basili 1996). There is also a problem with resistance to participation and measurement (Johnson 1997).

In discussing what makes good software engineering research, Shaw (2002) in examining papers at the International Conference on Software Engineering notes that “empirical models backed up by good statistics are uncommon”. She explains that
most papers represent a new procedure or technique and the research is either embodied in a tool, or simply described through the paper. Other papers tend to be “analytic models [which] support predictive analysis… [or] descriptive models [which] explain the structure of a problem area or expose important design decisions” (Shaw 2002).

Shaw (2002) lists the type of validation techniques as: analysis (experiment based), experience (real world use by others that demonstrates effectiveness, usefulness or correctness), example (single instance, toy problem based, etc), evaluation (feasibility study, pilot study), persuasion, and blatant assertion. In this section we present some of the metrics used for analysis, followed by a discussion on the use of case studies which provide further analysis as well as experience data (based as Shaw explains on experience independent to the researcher). Also discussed in this thesis are a few cases of example (where I have applied the techniques myself as a pilot), though this is present as a minor point and not the main basis of evidence in this research.

Some metrics used for analysis:

<table>
<thead>
<tr>
<th>Item</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adoption of various process descriptors</td>
<td>Count of participants who say they use it (validated through triangulation with observations and interview questions)</td>
</tr>
<tr>
<td>Usefulness of tools</td>
<td>Relative score on a 5 point scale</td>
</tr>
<tr>
<td></td>
<td>This is a rating by participants using a 5 point system, then scaled so all students contribute an equal weight over all, but it is divided between the tools according to their preference. (Validated through open questions in surveys and interviews)</td>
</tr>
<tr>
<td>Quality of software</td>
<td>Qualitative analysis focussing on:</td>
</tr>
<tr>
<td></td>
<td>• Ease for future reuse</td>
</tr>
<tr>
<td></td>
<td>• Adaptability</td>
</tr>
<tr>
<td></td>
<td>• Researches perception of room for improvement</td>
</tr>
<tr>
<td></td>
<td>Reuse score</td>
</tr>
<tr>
<td></td>
<td>This was taken on a ten point scale from both students and</td>
</tr>
</tbody>
</table>
## Table 2 Metrics used in this research

<table>
<thead>
<tr>
<th>Impact of SERE</th>
<th>Change in marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>The delta metric (described later) measured self improvement between students base line before the project and their project score.</td>
<td></td>
</tr>
</tbody>
</table>

**Qualitative analysis**

Students’ perspective examined through interviews

<table>
<thead>
<tr>
<th>Participation cost</th>
<th>Comparative ordinal data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students ratings on whether participation was more, the same, or less effort than expected, more, the same or less than past work they’ve done, more, the same or less than they felt it saved.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Participation benefit</th>
<th>Comparative ordinal data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students ratings on whether use of software engineering, documentation, etc was more, the same, or less effort than would otherwise have occurred. Separately students rated the benefit of this software engineering and documentation.</td>
<td></td>
</tr>
</tbody>
</table>

A large amount of data is also gathered through surveys (which include the use of binary answer questions), and some of this data has been used quantitatively as summary statistics.

In using the case study approach in this work, we followed the Guidelines as outlined in Kitchenham, L. Pickard and S. L. Pfleeger (1995) when planning and conducting the case studies. We have also taken other research into account and this is shown in our case study plans which provide extra summary information of the approach.

The case studies are presented in terms of:

**Aim:** What aspect of the over all study is being examined

**Type:** This is the type of case study based on Robert K. Yin’s classic classification of case studies (Yin 1994).
BSH type: This is a type according to the qualitative software engineering framework in the landmark paper by Basili, Selby and Hutchens (Basili, Selby et al. 1986).

Unit of Analysis: This is the subject of the investigation. To ensure more rigour various subject have been separated out into different case studies.

Embedded unit: This is a sub-unit of evaluation, i.e. the unit of analysis if looked at in a more granular way (in addition to it’s holistic way) is made up of these subunits.

The BSH type is a classification according to the qualitative software engineering framework in the landmark paper by Basili, Selby and Hutchens (Basili, Selby et al. 1986). BSH types are defined based on the number of teams replicating each project, and the number of different projects examined.

**Blocked subject project:** Many projects each of which is examined by many teams

**Replication Project:** One project which is examined by many teams

**Multi project variation:** Many project examined by the same team

**Single Project:** One project examined by one team.

Basili (Basili 1996) classified the later two types of cases as “quasi experiments”. They are more useful for large projects (due to lower running cost) and can be done *in vivo* making full use of experts. Generally interviews and qualitative analysis are used with these approaches. The down side is that as the name suggests, they are less rigorous than Blocked subject-project case studies and replication project case studies which are by definition controlled experiments and produce higher statistical confidence. Blocked subject project studies as well as replication studies meet the requirements of a formal experiment. These may be referred to as either case studies or experiments (Kitchenham, Pickard et al. 1995).

The specific case studies and experiments used in this research are presented in the methods section (pgs 96, 104, 109). While hard to provide as a list of metrics, various themes were examined in the case studies (as explain in their individual design) and
much of the collected data (including interview transcripts) was entered into the NVivo software package and “coded” (that is, sections of text were assigned to categories for use in later analysis). This approach is more commonly used in the social sciences, and our use of it was checked with Prof Nigel Fielding, a domain expert in the area.

Notes on the experimental approached used for the interviews and surveys can be found in the appendix, along with a set of the codes used to tag the data (see pg 313).

2.2 Improvement in the Research Environment

The Research Environment is significantly different from that of industry. Not only are the conditions and constraints different, particularly in a university (the environment this work focuses on), but the goals and reward system are different too. Researchers do not create “production software”, they do not work in a “production environment” and following a “production process” makes little to no sense. In understanding the research environment and its needs we can begin to appreciate what an improved process in the research environment could be expected to deliver.

2.2.1 Software Engineering for Research

Glass (2002) claims “computer science academics looking for… the one true approach to build software systems, that can be applied, universally, to any and all software projects” he juxtaposes this with software practitioners who each time they are presented with a solution cry out “…but our project is different”. In as much as computer science academics are themselves practitioners (building software for the purpose of their research) they too regularly claim “…but our project is different”.

The claim is made, that research is, by its nature, so radically different that software engineering practices simply do not apply. In our previous work we have rejected such claims (Oboler 2002) and argued instead that many of the current software engineering approaches are not appropriate to the needs of researchers – new tools, methodologies and (dynamic) combinations of existing software engineering practices are needed. Glass suggests the solution is greater appreciation for “ad hoc” approaches. In computing, “ad hoc” is defined as “contrived purely for the purpose in hand rather than planned carefully in advance” (Howe 2005). We see no reason why a purpose specific approach and careful planning are mutually exclusive – outside of
computing the definition of *ad hoc* does not say anything about the approach to planning. The Latin root of *ad hoc* means “to this”. An approach can be planned “to this” specific development process. To retain an “engineering like” element, we need only enable the planning to proceed in a structured manner, noting existing processes and tools that can be reused in this “ad hoc” approach.

Glass (Glass 2002) warns that “the solution approach must take into account the nature of the problem at hand.” He is derisive of many so-called meta-approaches, “collections of those problem-specific approaches with a problem-independent façade” (Glass 2002), and laments that as soon as a researcher starts work on a tailored approach to the problem at hand, someone comes along and proposes a “meta” approach to these tailored solutions. An approach for the research environment needs to therefore be more than just a collection of tools made available from industry.

Prior to our own research conducted in 2002 (Oboler 2002; Oboler 2003; Oboler, Squire et al. 2004), little work had been carried out investigating the use of software engineering by academic researchers. One unpublished paper was found describing the efforts at process improvement in the Computer Science Development Lab (CSDL) in Hawaii around 1992. This research was not continued and the reasons why are presented in this section. Other research found in the area included an analysis of existing research in which Glass, Ramesh and Vessey (2004) compare the topics, approaches and methods used in Computer Science, Software Engineering and Information Systems research and follows similar earlier investigations into the disciplines by the same authors (Glass, Vessey et al. 2002; Ramesh, Glass et al. 2004).

The remainder of the section summarises the Use of Software Engineering in Computer Science Research (USE CSR) project we conducted through 2002. The project investigated the lack of software engineering use in the research environment and the reasons why researchers found existing approaches unsuitable. The resulting recommendations for an improved software engineering methodology tailored to the research environment are present at the end of this section. The current research project fits within this framework and develops the part of this methodology that is aimed at initial researchers.
2.2.1.1 The CSDL Engineering Environment
Outside of the USE CSR work we have found a single research paper on software engineering for the research environment. A 1997 Technical report by Philip Johnson makes a number of claims based on the experience of his work conducted circa 1992. The technical report described CSDL’s engineering environment the focus of which was “to better understand and improve our process and product engineering within our academic research development environment” (Johnson 1997). Unfortunately the paper never made it beyond a technical report because this was not a research interest of the author, but rather a practical one and also because the author did not feel their experiences were significant enough for a journal/conference paper (Johnson 2004).

Findings of the technical report included the observation that volatility in research projects is of the form:

code > tests > class interfaces >> class hierarchy >> architecture

where architecture is most stable and code changes regularly. They found that it was reasonable to create external documentation for relatively stable work products such as architecture, yet completely unfeasible for things like code, tests, and class interfaces. “While documentation of these volatile components was absolutely essential to enable review, improvement, and reuse of these work products, the only feasible way to maintain accurate documentation in this highly exploratory environment was to generate it by parsing the primary work product (typically Lisp source code)” (Johnson 1997). They also found regression testing important and developed guidelines for this. The guidelines can be summed up as recommending incremental development of completely automated tests with only a minimal attempt at breadth.

2.2.1.2 Software Engineering in the University Research Environment
After finding a lack of any published work on the topic, the USE CSR project (Oboler 2002) surveyed researchers across the US and Australia, conducted case study investigations as well as a number of interviews to investigate the use of software engineering in the research environment. The results indicated a lack of use of software engineering within the academic research environment. More pointedly, it showed that many academics did not see the relevance of software engineering to
them, despite working in jobs where software was one of their primary outputs. A few were using industry-style software engineering and expending a vast amount of overkill on their projects.

### 2.2.1.3 Adapting other Software Engineering approaches

Through analysis of data collected in the CSE CSR project, it was found that software engineering as practised in industry, and taught at university, was seldom appropriate for research work. While many of the tools and techniques from software engineering were valuable, many were too heavy handed, time intensive or resource-intensive for the research environment.

This problem was exacerbated by the fact that most research ideas don’t work. As Bertrand Meyer, inventor of Eiffel and one of the software engineer experts interviewed put it, “the exception is that something works. Most ideas are stupid” (Meyer 2002). Unlike an industrial project where the goals are fairly clearly defined at the start, research projects are usually much more flexible, mutate rapidly (as new ideas occur to the researcher), and involve a team small enough that critical information can (at least for a time) exist in someone’s head and not need writing down. Without software engineering however, this information is eventually lost. The converse was those researchers using the overkill method (industry style software engineering) and spending vast amounts of time documenting ideas and modules that were later discarded.

### 2.2.1.4 Separation of Research and Development

USE CSR data also highlighted a problem resulting from the fact that most research software is written by students (usually PhD students) who are at the university for a limited time. In that time they need to complete their project and thesis as well as papers. There is no reward for the developers if they build a readable, maintainable, well-documented piece of code. On the contrary, funding is not provided for this and time spent on this is unproductive time in terms of university outputs. The same applies to code built by most other university researchers.

The solution, as we explained in the introduction (pg 3), is to encourage researchers to do a minimum amount of software engineering that is of fairly immediate benefit to their project. The rest of the software engineering effort, or more specifically re-
engineer effort, needs to come later and be shifted away from research staff, much as the preparation work in any of the physical sciences is shifted away from research staff to laboratory assistants.

While this thesis presents ideas on the software engineering that is of benefit to researchers, it is worth spending a moment considering the next step of re-engineering. Re-Engineering is “the systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance or evolvability at a lower cost, schedule or risk to the customer” (Tilley 1995). In a RESET process we would be particularly concerned with quality improvements in evolvability at lower costs, on tighter schedules and with less risk to the customer, who we define as potential researchers who could reuse this work. While the re-engineering will not itself be examined in this research, one of the key aims of process improvement in the research environment is to make this re-engineering easier – with as little effort as possible from the initial researcher, except where such effort has worthwhile benefits for the researcher as well. This is examined in more detail in section 9.1 (pg 183). For the engineer two key tasks are likely to be common parts of the re-engineer process, they are restructuring and reverse engineering. They can be explained as:

**Restructuring** involves removing unused code, renaming variables, and updating code to meet any needed standards. The purpose is to counter the law of deterioration (Lehman 1980). Tools exist that enable some of this work to be semi-automated, e.g. renaming of all instances of a variable (Griswold and Notkin 1993).

**Reverse Engineering** involves recovering knowledge about how a system works. This can be in the form of design recovery (creating an abstraction of how a system works from what it does) or re-documentation (documenting the existing structure of the system). While tools are considered useful for re-engineering, a hybrid approach involving both human expertise and automatic tool assistance for the reverse-engineering part appears to be the best strategy (Klösch 1996).
While this section has provided some insight into the RESET process it should be stressed that this process itself was not included in the present research, only the steps leading up to it have been considered.

2.2.2 Legacy properties in the Research Environment
The problems of the research environment are not all unique. Some of these problems bear similarity to the problems affecting legacy systems. This section highlights the comparisons between legacy systems and research, and the possible lessons we can learn from the legacy system experience.

2.2.2.1 Legacy Systems
“Legacy system” was informally defined by Bennett as “large software systems that we don’t know how to cope with but that are vital to our organisation” (Bennett 1995). Nicolas Gold extends this by adding that they comprise “critical software that cannot be modified efficiently” (Gold 1998). One of the key problems with legacy system is that they “have their own legacy data stores with their own designs that are implemented using a variety of technologies” (Ambler 2000). Legacy systems are associated with high maintenance costs, the result of “the degraded structure that results from prolonged maintenance” (Warren 1998) (pp4-9) and contain “large amounts of implicit and undocumented information and knowledge”(Klösch 1996). According to Ian Warren (Warren 1998) the problems arise as:

- Maintenance tasks are assigned to junior staff, as maintenance has a poor image
- Legacy Systems have often been developed for efficiency rather than maintainability, this makes understanding difficult
- The impact of change is hard to predict when the system is in a structurally degraded state due to many changes made by different people
- Documentation suffers due to budget and resource constraints and is often inconsistent or out of date as a result
- Many legacy systems are developed according to ad-hoc techniques rather than engineering principles which show technical maturity.

2.2.2.2 Research Software comparisons
Research software can be described as exploratory in nature. Where one project is a continuation of another, or tries to build on past work, the previous work can be
regarded as “critical software”. At present Research Software often cannot be “modified efficiently”. It is clear from experience that such software almost always uses custom data stores and a variety of technologies. Like legacy systems, research software by its nature is maintained (usually by adaptive maintenance) by different people at different points in time. Research software is usually developed along the lines of the “Big Ball of Mud” which “represents a triumph of utility… [as] workmanship is sacrificed for functionality. Durability can be sacrificed as… an incomprehensible program defies attempts at maintenance” (Foote and Yoder) this is very similar to the general incomprehensibility of old legacy systems, despite the youth of the research software.

Inefficiencies in research systems are often a result of:

- Assigning maintenance tasks to junior researchers, as maintenance is a poor use of more senior researchers time
- Research code is often developed for efficiency of production (and occasionally of execution) rather than maintainability
- The impact of changes is hard to predict when the system is in a structurally degraded state, due to many changes made by different people
- Documentation suffers as a result of resource constraints, particularly as producers of code usually have writing up priorities towards the end of the project. Code used by multiple people will often have inconsistent or out of date documentation – if it has documentation at all.
- Development is according to ad-hoc techniques rather than engineering principles.

While research systems are typically very small by legacy system standards, and are typically developed by very small teams or individuals compared to the large teams for legacy systems, many of the problems are startlingly similar – particularly when we consider research code that is reused for other projects.

2.2.2.3 Solutions to the legacy problem

Ian Warren (Warren 1998, p10) suggests a replacement for the legacy systems is an “evolutionary system [which] is capable of accommodating changes over an extended operational lifetime”. This requires that development uses established management
techniques, recognises pre-planned product improvement, captures requirements effectively, support rapid accommodation for change (including reuse and user configurable systems), makes effective use of existing assets (manage related products together and encourage reuse), and develops information rich systems (collects information on why and how a system was changed). In order for an SDLC to be useful in academia, it needs to be “evolutionary in nature” (Pressman 2002), the problem is that many of the suggestions clash with the nature of the research environment (where not only can new features be added, but large ideas and their related code are often discarded) and priorities of researchers (to publish, write-up etc rather than perfect code). In short, the burden placed on those creating “evolutionary systems” to replace legacy systems is far too great to be placed on researchers.

To make the lessons from legacy systems useful in the academic environment, without compromising the priorities of researchers, a significant part of the burden of evolution needs to be transferred. The RAISER / RESET SDLC (presented in the introduction, pg 4) provides a software development life cycle based on the principle of a separation between research and stabilisation.

2.2.3 Thoughts on Process Modelling
Ward, Fayad and Laitinen (Ward, Fayad et al. 2001) suggest that “abstract side-by-side comparisons of process models are essentially meaningless” as they are “out of the context of the particular group’s particular circumstance”. While it is the nature of scientists to try to extrapolate the general from the specific, this extrapolation needs to be valid.

In his thesis Thunem (Thunem 1997) rejects the inclusion of environmental variables in the definition of process as being “too broad”. Such a broad definition makes it impossible to abstractly compare processes side-by-side – unless they occur in exactly the same environment. According to both (Ward, Fayad et al. 2001) and (Glass 2002) this would appear to be a positive development – unless one is focused on modelling rather than on process improvement as with Thunem (Thunem 1997).

We accept that each research process is unique, and while tools may be of general use, each researcher needs to find the best set of tools for them and their current project.
Attempts to find a meta-solution in the best combination of tools are in our opinion doomed to fail, unless targeted at a very specific problem class – this is not our aim in this research but remains as an open problem for experts to explore in their own research fields.

2.2.4 Measurements of Success
In an experimental intervention that aims to provide better approaches, the experimental subjects determine success, choosing methods and tools that work for them, or rejecting the experiment outright. If a new approach makes researchers feel their lives have been made easier, their risks reduced or their effort more focused, this is sufficient to demonstrate an improvement in the process of their research. While an improvement for one researcher may be random success, a high degree of acceptance would indicate approaches of general usefulness. Where adoption occurs and positive feedback is provided, or negative feedback about problems stop, this would indicate a successful intervention. Another example of success is when researchers collectively move to work on, or complain about, more fundamental problems.

The continually evolving approach we made available in this work is another measure of success as deeper insights are made into the nature of the problem and approach to a solution. The success of this insight is of necessity left to the reader.

2.3 Core process concepts
2.3.1 The Nature of a Process
An elementary notion of process is the systematic approach to creating a product or completing a task (Osterweil 1987). Dowson defines software processes as the steps “for creating, maintaining and evolving [a] software system” (Dowson 1993). Such definitions limit the activities examined to those carried out by people. A more open definition is that of Osterweil who defines process as “any mechanism used to carry out work or achieve a goal in an orderly way” (Osterweil 1987). He includes not only the way software developers go about doing their jobs, but also the tools, guidelines and methodologies they use doing it. This view is similar to (Lonchamp 1993) who defines process as “a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints...”. In his thesis on process modelling (Thunem 1997) rejects these non-
elementary definition as “too broad”, desiring a separation between tasks and their environment. We believe this to be a simplification to make the modelling task easier. While the environment around a house may have little impact on an architect’s plan, it can have a very real impact on those trying to live there. For the purpose of this research which examines process improvement in the research environment and seeks to influence and improve it, we will use a broader definition and define process as: a set of steps, with related artifacts, tools, resources, organizational structures and environmental, personal and subject based constraints.

2.3.2 Defining Process Improvement

Process improvement is optimisation of the way a task is done. A distinction can be made between cases where the task is exactly repetitive (e.g. mass production) compared with more creative processes (e.g. design). Exactly repetitive tasks aim to create uniformity in the product while creative tasks aim for continual improvement in the product or the creation of slightly or drastically different products.

Originally, process optimisation was introduced for exactly repetitive tasks. The most common metrics were efficiency and quality. To be more efficient means producing more output with fewer inputs. This could relate to a reduction in waste products (more efficient use of raw materials) or to doing the job within a shorter period of time. An improvement in process quality means systematically reducing the number of rejects (or defects). This is largely achieved through automation, the extreme case being full automation (including mechanisation and computerisation) of a deterministic process. For non-repetitive tasks (research by its nature being an extreme case) the number of additional metrics is far higher as there is more room for change. In both cases specific metrics must be selected for optimisation.

The difference between exactly repetitive tasks and more creative tasks can be explained through an analogy: - An old-fashioned printer with a printing press would learn and improve over time. Given the same text and asked to create new plates a few years later the output would be different and one hopes better. The changes are due to the creative element in the task of layout. By comparison, a computer’s printer reproduces the same output exactly each time the print job is run. There is still a creative layout element, but it has been isolated from the printing process. The second
printing process we discuss is now automated and an entirely predictable and repetitive task.

Full optimisation for one metric may cause lower performance in other metrics. Process Improvement therefore seeks to find the right balance between completing goals. It can be seen as a multivariable optimisation task subject to a set of constraints.

2.3.3 Meta-Concepts: the Process of Process Improvement
The process of process improvement is a meta-research concept. If we see a research project as a task, we can define the work of a researcher trying to complete that task as a process. The meta level involves an examination of how the researcher goes about their work. When we create ways of helping researchers, our work too can be regarded as a task. It then becomes possible to discuss our own process (of offering researchers assistance) and to systematically examine how this process (of offering assistance) can be improved. This meta-meta level is (for reasons explain later in this thesis) the highest conceptual level at which one can work.
3 A Review of Past Developments

3.1 Development of the Process Improvement paradigm

This section discusses the development of the Process Improvement paradigm up until the point where it merges with the use of technology as an enable of change.

3.1.1 Taylorism and Scientific Management

Process Improvement as an engineering discipline started in the late nineteenth century when the attention of some engineers was being diverted from the engineering of physical systems to the engineering of human systems (Jacques 1997). One of these early engineers was Frederick W. Taylor who conducted a series of “time and motion studies” at the Bethlehem Steel Company. Taylor decomposed a worker’s job, analysed the motions involved, altered them, and retrained staff to use the new process. For one type of job this led to an almost 400% increase in the productivity. Taylor’s philosophy became known as Taylorism.

Through the Second Industrial Revolution (through to 1950) the Scientific Management Movement, and particularly Taylor’s “The Principles of Scientific Management” (1911) had a strong influence on ideas and approaches to redesigning business processes. The manufacturing sector was focused upon and targeted for efficiency gains (Kock 2001). While Taylorism eventually led to job dissatisfaction by workers, and a resulting decrease in productivity, it also left a lasting positive contribution. Work places were (as a result of Taylorism) redesigned for efficiency and process improvement research gained momentum.

3.1.2 The Humanist School and the Hawthorne Effect

Critics of Scientific Management, like Elton Mayo, shifted the focus in process improvement to people and gave rise to the humanist school of management (Draper 2005). In a set of experiments at the Hawthorne works of the Western Electric Company, experimenters varied a number of factors over time, always showing an improvement in worker productivity. This held true even when conditions were systematically reverted to their initial state (Mayo 1933). The results, known as the Hawthorne effect, showed that experimenters were improving productivity by
focusing on workers and making them feel important. It disproved Taylor’s assumption that financial reward was in all cases the chief driving factor for workers. Mayo presented the idea of the team and the social context of the work place (Mayo 1933; Draper 2005). In the later years of the Industrial Revolution the humanist school of management became dominant and shifted the focus largely from process to people.

### 3.1.3 Total Quality Management (TQM)

Total Quality Management makes the workers themselves (individually and collectively) responsible for quality. Under TQM, quality is checked at every stage and throughout the company. As workers do quality checks themselves, they are empowered. It reconciles the idea of incremental improvement and the human element. TQM was introduced after WWII, first in Japan (building it up to economic superpower status) and from the 1980’s in the US and the West. The focus was still largely manufacturing. However, as quality was hard to define, over time TQM became more focused on Quality Processes (Kock 2001). This in turn led to a focus on certification and a bureaucratisation of the process (leading back to disempowerment of workers).

### 3.2 Emergence of Technology as an enabler of change

Computer Science is the discipline of solving problems through the use of computers (Comer, Gries et al. 1989; Burch 1999; University of Oxford 2002). In recent times hardware has usually served as a generic platform, while the software provides the specific application to solve problems. Historically this was not true. The history of technology as an enabler of change starts pre-software, with the creation of the computer itself and its use in problem solving.

In this section we look at the early history of computer science as a means of process automation, followed by developments as computing becomes more “engineering—like” in response to the software crisis. As part of the shift to software engineering, software development begins to be modelled and then managed as a process and various Software Development Life Cycles are proposed. Software engineering research becomes the discipline of improving the way people go about developing software. i.e. improving the software process.
3.2.1 The Rise of Computing: Improvement Through Automation

In 1804 Joseph Marie Jacquard invented the Jacquard loom, a device originally conceived for automating the process of weaving silk. The automated loom was the first use of punch cards to issue instructions in a mechanised process. It allowed a single operator to do the job that previously required a team of weavers.

In 1822 Charles Babbage, a professor of mathematics at Cambridge, invented the Difference Engine in order to automate the creation of star charts for navigation at sea and other vital tables. Babbage intended to improve the process of table generation both by increasing efficiency and by reducing the error rate. The Difference Engine concept led to the Analytical Engine and from there to the Hollerith Tabulator (used to improve the process of compiling the census data of 1890) and the rise of the Tabulating Machine Company, now known as International Business Machines or IBM (IBM 1953; Tuck 2002).

Babbage's Analytical Engine was the first “universal digital computer” (Turing 1950) and, although mechanical, it had many of the features of a modern computer (Copeland 2000; Charles Babbage Institute 2002). The need to improve processes and specifically to automate or computerise them has been a driving force behind technological advances and improvements in computer science.

A key aspect of process improvement through automation is determining which processes can be automated. Work on the Halting Problem led Alan Turing to the development of the abstract concept of a Turing machine (Turing 1936). This machine is the basis of the modern theory of computation and computability. It also led to the development in 1943 of Colossus, the first digital, electronic and partially programmable computer. Colossus was the key to improving the process of breaking the German Lorenz ciphers during World War II (Zorpette 1987). The use of machines (Colossus and others) was said by Gen. Eisenhower to have “saved thousands of British and American lives and, in no small way, contributed to the speed with which the enemy was routed and forced to surrender” (Eisenhower 1945). The Colossus computer was “the chief hidden factor that helped the Allies win" (Kahn 1991). The true impact in process improvement that came about due to Colossus specifically is hard to determine. The machine itself was highly classified and
continued in operation long after the war ended while the allies perpetuated the myth that Lorenz was unbreakable.

The ENIAC (operational from February 1946) is the world's first general purpose electronic digital computer (Weik 1961; Winegrad and Akera 1996; Oldehoeft 2000). The ENIAC patent (US patent No. 3,120,606) shows that from the start computers were primarily intended to automate data processing for scientific work. An improvement in the process was thought to be of great benefit to science. The patent notes that “the most advanced machines have greatly reduced the time required for arriving at solutions to problems which might have required months or days by older procedures. This advance, however, is not adequate for many problems encountered in modern scientific work and the present invention is intended to reduce to seconds such lengthy computations” (Eckert and Mauchly 1947).

From here machines became programmable, and the history of process improvement shifts to software, first as the means of improvement and then as the part of the development process to be improved.

**3.2.2 The Shift to Software Engineering**

As early as 1950 Turing (Turing 1950) noted that for artificial intelligence research to succeed, improvements in both programming and engineering were needed. The same was true in many other areas.

The “software crisis” highlighted the inadequacies with the then current process of producing software. These were discussed at the first conference on software engineering, in 1968 (Naur and Randell 1969). A year later, a second conference focused on making software development more “engineering-like” (Randell and Buxton 1970).

The first systematic approach to improving the entire software process, the software development lifecycle (SDLC) as it became known, was introduced in 1970 by Royce. In the Waterfall model (Royce 1970), Royce tried to map out the steps to successfully tackle the process of producing large-scale software. He first presented a two-step approach: “analysis” which led to “coding”. Royce explained that for small projects in
which the software would only be operated by the developers, this would be sufficient. He then introduced a process improvement which involved breaking the development process down into certain essential steps. To improve the process in large systems, he argued, these steps would need to be followed.

In 1988 Boehm introduced a new pattern for process improvement in the development of software - the Spiral SDLC (Boehm 1988). He suggested a new model was needed to increase reuse. Unlike other production processes, where a part is a physical object, in software the hard part is the idea and once coded it can be reused many times. The spiral model focuses on the evolving nature of software. It uses prototyping and repeated risk assessment phases as the process progresses.

In 1989, the ACM education board, endorsed a report outlining a computer science curriculum including “Software Methodology and Engineering”. This featured modular design, abstraction and lifecycles – all tools or ways of improving the software process. The course aimed to teach students how software can be designed for two metrics, understandability and modifiability (Denning, Comer et al. 1989). Reuse was not explicitly included.

In 1992 Krueger (Krueger 1992) noted that improvement through reuse had failed to become standard practice in industry. In 1996 Devos and Tilman (Devos and Tilman 1996) noted that straightforward OOA/OOD focused on reuse and evolutionary needs too late in the process.

Robillard and Robillard (1998) compared student development work with industry developed work. They showed that university work was dominated by the programming phase. Humphrey (1988) arrived at the same conclusion, adding that undergraduate students failed to use software engineering practices i.e. to take a process approach to development, unless specifically directed to do so. The students claimed that class projects were too small to need Software Engineering. Humphrey described the common student ethic as “ignoring planning, design and quality in a mad rush to start coding”. This is a product rather than process driven approach.
Cook, Ji and Harrison (2000) suggested that repeatability might be less relevant for longer term process improvement in software engineering than in other engineering fields. They suggested the process should focus on design activities and the adaptability of the software development process. They discuss the metric of evolvability, which is based on code quality, use of an evolution process, and the organisational environment in which development takes place. Cook et al. recommended four steps to software evolvability: analyse which parts of the system might need to change, implement the change, re-stabilise the product to remove any introduced errors, and test the changed product.

Past research has suggested the need for process improvement across the computer science field, from industry to education (Steier, Coyne et al. 1993; Humphrey 1998; Robillard and Robillard 1998; Cook, Ji et al. 2000).

3.3 Convergence of change and process improvement

On the one hand we have technology being used to enable change, and on the other we have a push towards process improvement. With the adoption of Business Process Re-engineering by many organisations, technology became a key driver in a particular form of organisational process improvement. The CMM was created to provide an assessment of developers capabilities in building software. CMM evaluation was based on a review of processes using a score card. It was soon being used to improve the process themselves in order to obtain better scores. The developers of CMM were in favour of continual improvement rather than the score card approach they’d created. ISO9000 provided international certification that a company’s procedures meet certain minimum standards. ISO9000 pushed the scorecard approach from which CMM had tried to distance itself. All three approaches required software companies to document and assess their processes. This enabled reflection on process improvement.

While BPR, CMM and ISO9000 targeted large companies, other approaches such as the Personal Software Process (PSP), Team Software Process (TSP), PSP/Baseline, and Process for Improving Programming Skill in Industry (PIPSI) were developed to adapt the process approach to smaller groups. Some of these approaches are now discussed in more depth.
3.3.1.1 Business Process Reengineering (BPR)

Business Process Reengineering (BPR) was a process improvement technique that analysed how the business worked and then replaced some work practices with a new way of doing things, often “best practice”. It was often facilitated by the introduction of an ERP system. The shift to BPR was itself a response to the shortcomings of TQM under the growing impact of information technology and a more rapidly changing business world (Davenport and Short 1990; Hammer 1990; Davenport 1993a; Davenport 1993; Hammer and Champy 1993). Unlike TQM, BRP’s solution was a once off radical change (Nickols 1993; Malhotra 1998; Al-Mashari and Zairi 2000). BRP became the mainstream approach from 1990 (Malhotra 1998; Kock 2001) but largely fell out of favour in the mid 90s as business and consultants shifted focus from process improvements to cost savings (Davenport 1995; Malhotra 1998).

In the UK uptake of BPR was more cautious. A study of articles mentioning BPR in the Financial Times showed a peak from 1994 to 1995, which under analysis showed first praise of BPR and then sharp criticisms (Graham, Lloyd et al. 2000). In the UK, BPR was mostly used as a buzz word by vendors and consultants to sell existing products or knowledge bases. By 2000, while many of the consultants had moved on, companies were still finishing their BPR initiatives (Graham, Lloyd et al. 2000). Business Process Improvement as a more general tool has re-absorbed BPR. The report on BPR in the UK concluded that “post-BPR, process continues as a central theme in management discourse and practice” (Graham, Lloyd et al. 2000).

3.3.1.2 ISO-9000

ISO 9000 is an internationally recognised set of standards for establishing quality. They are developed and maintained by the International Standards Organisation. The most common use of ISO 9000 is for certification purposes, allowing developers to demonstrate their competence.

Wang 2002; Praxiom 2004). We will focus here on ISO IEC 90003:2004 as a simple and relevant approach to ISO9000. Given the nature of this thesis we will specifically focus on the process improvement requirements, rather than those related to certification.

From the summary of ISO90003 (Praxiom 2004) we can see that a small number of process improvement steps have been included (the full list of steps runs to well over a hundred items). The system referred to in Table 3 is the Quality System i.e. the system responsible for monitoring and improving the quality of both processes and products.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Identify the processes that make up your system and improve the effectiveness of your processes.</td>
</tr>
<tr>
<td>5.1</td>
<td>Improve your system by performing quality reviews and providing resources to implement improvements</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Plan how you're going to improve the effectiveness of your system</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Evaluate the effectiveness of your system</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Examine the inputs to management reviews and opportunities to improve</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Generate outputs from the management reviews including actions to improve the system’s effectiveness as well as the software products</td>
</tr>
<tr>
<td>6.1</td>
<td>Provide the resources needed to improve your system.</td>
</tr>
<tr>
<td>8.1</td>
<td>In planning and implementing remedial processes, use monitoring, measuring, and analytical processes to continually improve the effectiveness of your system</td>
</tr>
</tbody>
</table>

Table 3 ISO90003 summary

Processes are identified and evaluations of various types are carried out. Process improvement is not seen as pervasive but rather as occurring at discrete points, largely as a by-product of other activities. Requirement 8.1 requires that when the improvement is a system correction, it takes place in an “engineering-like” manner. ISO 9001, even with the ISO90003 guidance specifically tailored to the Software Industry, clearly focuses on quality assurance. It can allow certification when only minimum effort is applied to incorporate process improvement (Thunem 1997). It is seen as describing the minimum criteria for a reasonable quality management system, but fails to cover the breadth of process improvement needs (Paulk 1995; Thunem 1997). Further guidance is provided in ISO 9004:2000, but this is not mandatory for certification.
ISO 9004:2000 focuses specifically on process improvement (Zimmermann 1980). It is a set of guidelines and recommendations, and is not intended for certification, regulatory or contractual use, nor is it a guide to assist in ISO 9004 implementation (The ISO Standards Bookshop 2000; Balkwell Consulting 2005). ISO9004 exists to address the gap in ISO9001 and provides the additional guidance needed to enable process improvement. It is often overlooked by users (Zimmermann 1980) and very little research discussing it exists (7 results in the IEEE archive compared to 137 for ISO9001). This is most likely a result of its non-mandatory nature. A comparison between ISO9001:2000 (including ISO9004) and CMMI, along the lines of Paulk’s work (Paulk 1995) would be useful.

3.3.1.3 Capability Maturity Model (CMM)
CMM was originally developed in 1987 (Humphrey and Sweet 1987) as an assessment model to be applied to software providers. It provided five levels of maturity for a software company ranging from an initial state through to a continually optimising state (see Figure 4). The maturity rating indicates a level of risk should a project be assigned to that company. The higher the CMM rating the more attention the company pays to process improvement.

![Capability Maturity Model (CMM)](image)
CMM is based on the belief that repeating proven methods of a known and mature software process will consistently give good results. It is acknowledged that in undisciplined organizations some software projects produce excellent results, as a result of “the heroic efforts of a dedicated team” (Paulk, Curtis et al. 1993). CMM aims to replace this with consistent and reliable results. In the last few years CMMI, which builds on the CMM framework and provides products for integration and evaluation, has become popular. The underlying principle of CMMI is however still the CMM’s five levels of maturity. CMMI allows a wider range of issues to be considered, but is still focused on a plan driven approach to minimize risk; an approach that makes little sense in the research environment where “risk” is pervasive.

The levels of CMM
The following is taken from (Paulk, Curtis et al. 1993).

CMM Level 1 - Initial
The development process is ad hoc. The success or failure is dependent upon individuals and the effort they apply. There are few if any defined processes. (Paulk, Curtis et al. 1993)

CMM Level 2 - Repeatable
The development process includes basic project management such as cost, schedule and functionality (Paulk, Curtis et al. 1993). Where the company has undertaken a similar project before, they should be able to deliver to a similar standard.

CMM Level 3 - Defined
The development process for all projects is based on the company’s standard development process and may then be customized as needed. The company’s standard development process documents and standardizes both management and engineering activities from across the company.

CMM Level 4 - Managed
The development process as well as the product is measured. Both product and process are understood qualitatively and this data is used to control the process.
CMM Level 5 - Optimized

Measurements of the process and product are used to enable continuous process improvement. New ideas are tried and the results recorded in efforts to find increasingly optimal processes.

CMM and process improvement

Once introduced, it was found that the development organisations could use the CMM model for internal process improvement (Paulk 1995). CMM has an explicit emphasis on continuous process improvement, an emphasis lacking in ISO 9001 (Paulk 1995). It also argues to include the environmental factors as part of the process (Paulk, Curtis et al. 1993).

CMM encourages continuous improvement. The CMM authors believed this can only occur through “focused and sustained effort towards building a process infrastructure of effective software engineering and management practices” (Paulk, Curtis et al. 1993). This is a combination of a process-based view and a systems perspective of the process that includes the development environment. CMM at its higher stages required demonstration of continual process improvement. In 1993 only 117 companies worldwide were certified to CMM Level five, and 67 of these were in India. (Braue 2003)

3.3.1.4 The Personal Software Process (PSP)

The Personal Software Process (PSP) (Humphrey 1994a; Humphrey 1995) was created at the Software Engineering Institute (SEI) at Carnegie Mellon University. It “scales down industrial software practices to fit the needs of small-scale program development” (Humphrey 1994a). The idea behind the PSP is that a defined and measured process will help software engineers improve their performance (Humphrey 1994b).

The PSP shows students and engineers “how to measure, estimate, schedule and track their work” (O'Connor and Coleman 2002) and is a cross between a course and “boot camp” (O'Connor and Coleman 2002). During the course students write ten programs while collecting and analysing their own process and product data. Students convince
themselves of the benefits based on their own experience. There are four levels to the PSP that students work through (see Figure 5).

**Figure 5: The four levels to the PSP**

**PSP 0**
Students establish their current practice and learn to measure it and the results.

**PSP 1**
Students learn to make size, resource and schedule plans and estimations.

**PSP 2**
Students learn and use defect and quality management (including reviews) and integrate them, noting the results on their performance.

**PSP 3**
Students learn to scale up the methods they are using for use in large projects.

There is an acknowledged problem with adoption of the PSP, and Software Engineers will not adopt it without experience. The course was created to teach people how to
use it and acts as an intervention, forcing students to use the PSP (in order to pass) while demonstrating the change in their own practices to them. One result of the PSP is a reduction in coding time and an increase in design and testing time (Humphrey 1998).

An underlying assumption in the PSP is that the improvement in performance will outweigh the cost of using the PSP (Johnson, Kou et al. 2003). Research however suggests, “it would impose an excessively strict process on [developers] and that the extra work would not pay off” (Borstler, Carrington et al. 2002) as a result 72 of the 76 PSP trainers in one class had stopped using it (Borstler, Carrington et al. 2002). Other results have highlighted the overhead in recording data and accuracy of the data recorded manually (O’Connor and Coleman 2002).

The use of a course as an intervention to change practice (temporarily) and demonstrate the result is a useful software engineering approach. While it may not change practise permanently, it will hopefully at minimum give an appreciation of quality.

3.3.1.5 PSP/Baseline
Experience at the Collaborative Software Development Laboratory (University of Hawaii) found that PSP is more suited to the research environment than CMM. This was due to the top down administrative support needed for CMM, and the assumption of a large institutional infrastructure (Johnson 1997). While PSP was seen as an improvement, it too had problems including; PSP is manual - data collection and analysis is a burden that the benefit does not justify, PSP is broad and comprehensive – there is too much attention to detail and this hinders adoption, PSP processes are hard-wired – a complete PSP is not suitable for all environments and nature of PSP suggests an all or nothing adoption (Johnson 1997).

PSP/Baseline was designed to allow each researcher “their own, individualized development process” (Johnson 1997). It addressed the problems of PSP by being fully automated, limited in scope, and highly customisable. PSP/Baseline was targeted at Bachelor, Masters and PhD students as well as research staff. A highlight of the
PSP/Baseline outline is the separation of “Instantaneous benefits” from “Continuous benefits” (Johnson 1997).

More recent research from the CSDL suggests that even the requirements of PSP/Baseline were too work-intensive. Barriers to adoption are: the cost of gathering data, quality of data (accuracy of metrics are often doubtful), and the lack of utility for the person gathering the data (Johnson 2001). In addition, students were found to context switch, and regarded the quality process as a separate task to software development. Many consider continual context switching far too intrusive and instead desire long periods of uninterrupted focus in order to achieve effective and efficient development. Outside of the class the additional tasks required for PSP were seldom carried out even with the computer support provided (Johnson, Kou et al. 2003).

The solution under development and experimentation since 2001 has been “Hackystat” (Johnson 2001), a software tool that allows ubiquitous collection and analysis of software development metrics. Hackystat uses “sensors” configured by the user and attach to their software tools to collect data unobtrusively whenever development occurs. Data is not stored in development phases (as this requires manual intervention) but rather in terms of file and tool accesses. This creates a different set of metrics to the PSP metrics, but it’s thought these can be equally useful.

### 3.3.1.6 PIPSI

The Process for Improving Programming Skill in Industry (PIPSI) seeks to create a freeware process improvement framework to meet the needs of the SME market in Europe (IPSSI Project 2002). It is driven from a bottom-up perspective and aims to be more attractive and useful to developers in the target environment (O’Connor and Coleman 2002). The three main elements are defining a personal process, personal project management, and personal quality management. Like the PSP, the process is based on measurement and the idea that if engineers keep these measurements they will see the benefit of engineering. The PIPSI framework aims to have engineers “understand the fundamental relationship between size and effort and, through this understanding, enable them to improve their estimating abilities”. It also incorporates personal code reviews and checklists.
Like PSP, PIPSI is based on a number of exercises. The amount of data collected is increased at each stage. PIPSI was trialled in 4 case studies, each involving a group of students in a different European country. Two of the groups involved 3rd year undergraduate students, one involved postgraduate students and the final involved industry participants (O'Connor and Coleman 2002). “In all cases PIPSI was tailored to the specific needs of the group being trained” (O'Connor and Coleman 2002). The case studies did not show an improvement in effort estimation, but they did show an improvement in size estimation and engaged students in thinking about development in new ways (O'Connor and Coleman 2002). The use of reviews was found to be the most helpful aspect of the program for those who chose to invest fully in the program (O'Connor and Coleman 2002).

3.4 Improving the Process of Process Improvement

With the wide spread use of BPR and classification under CMM and ISO9000, organisations developed a need to stay in front of their competitors. Improving processes was not enough. The processes needed to be continually improved to keep the company at the leading edge. This brought about a philosophy of continuous improvement – and the idea that process improvement was itself a process started gaining acceptance.

One of the originators of BPR noted in 1996 (as PBR was losing popularity) that the most important element to emerge from it had been a process orientated view, “reengineering and TQM are merely different pews in the church of process improvement” (Hammer 1996). This statement itself demonstrates the shift in focus from the discussion of approach to the abstract level of process and process improvement. By 2000 when consultants were moving away from BPR (Graham, Lloyd et al. 2000), at this about this time Ward et. al. (Ward, Fayad et al. 2001) reflected that “the term ‘process’ had been used almost exclusively in organizational development programs and human resource departments” but was at this point “on the verge of Internet startup buzzwordhood” (Ward, Fayad et al. 2001). He states that in professional opinion, if not in actual practice, companies had been convinced to adopt reasonable (process driven) development practices. The idea was not new, just ready for adoption. The concept of improving the software process by treating the process as
an object in its own right, had been around since 1987 (Osterweil 1987) when Osterweil recommended treating software processes as software themselves.

In this section we deconstruct this concept of treating the process as an object. A critical layer of understanding behind this thesis involves objectifying the development process as a whole, so it too can be analysed and systematically improved. The process of working on this new object is the process of process improvement. This idea originated with process programming and created the more general field of process modelling. In introducing this topic here we have abstracted and adapted the concepts to the context of the academic research setting. A discussion of the issues related to the adaptation is available in the next chapter.

### 3.4.1 Process Programming

In what was judged (Osterweil 1987) as “the most influential paper of ICSE 9”, Osterweil discusses how “software process are software too” (Osterweil 1987). He explains that “a process is a vehicle for doing a job, [while] a process description is a specification of how the job is to be done”. He explains that “the process description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior” (Osterweil 1987). Osterweil explains that the reason for objectifying processes is that “processes are hard to comprehend and reason about, while process descriptions, as static objects, are far easier to comprehend” (Osterweil 1987). This follows observations made by Dijkstra in his famous letter on the GOTO statement (Dijkstra 1968), namely that humans understand static entities (programs) far more easily than they understand dynamic entities (the run time processes created from these programs).

Osterweil suggests, “that contemporary ‘programming’ techniques and formalisms be used to express software process descriptions” (Osterweil 1987). If one believes in a process improvement approach to software engineering, the idea of applying existing software engineering knowledge to the task of improving our processes seems an ideal approach. Both the risk and learning curve are low due to knowledge reuse.

Osterweil also describes process programming as, “the activity of expressing software process descriptions with the aid of programming techniques” (Osterweil 1987). This
is a much more limited approach and far less useful than the broad outline. While Osterweil acknowledged that high level process descriptors are also useful as “general prescriptions for how software work is to be done” (Osterweil 1987) a larger focus unfortunately became the programming of low level processes. This is far too restrictive for the people involved and startlingly similar to a pure Taylorist approach. Osterweil recognised this and stressed the importance of the process programmer leaving some tasks less rigorously defined to allow the human practitioner room to act. It still seems like micro-management and entirely unsuitable for the research environment, and most other software development environments as well. We reject the low level descriptors as unhelpful, but adopt the idea of high level process descriptors as guidelines to a general approach.

At the implementation level, Osterweil recognised that there was not yet an ideal language for process programming. Since that time we have seen the development of design languages such as UML. While detailed UML diagrams would still be too much of a burden, the use of UML for high level architectural design seems reasonable for software development in the research environment, and may also be suitable for process design.

3.5 Summary
In this section we have discussed the development of process improvement from its industrial setting, the rise of the computer as a source for automation, and the convergence of these two drivers in approaches such as Business Process Re-engineering, ISO 9000, and the CMM. We have also discuss further developments as the improvement effort itself began to be scrutinised and targeted for improvement.
4 Discussion and development of key themes

In this chapter the key themes arising from the previous chapters are highlighted and discussed. Lessons are placed within the context of this research, and some new contributions building purely on the literature are introduced. These lessons provide additional foundations to the SERE approach and further position it within the context of both process improvement and the continual development of computer science as a discipline.

4.1 Building an Improvement Paradigm

Taylorism, the humanist approach, and Total Quality Management all provide lessons we can use in developing an improvement paradigm for the research environment. The main lesson is that, while research is clearly a process, it is not an exactly repetitive one. The research environment is influenced by many human factors, and while each researcher is responsible for their own work, quality of software is not particularly important. From this starting point and an understanding of what each of these statements means, it becomes possible to build an improvement paradigm that does indeed meet the needs of the academic research environment.

Taylorism as exemplified through the time and motion studies would be alarming to researchers, yet the idea of relative efficiencies and completing a task with the least amount of effort is critical to improving the research process. From Taylorism we take the idea of reducing effort through process automation for the exactly repetitive tasks in research. Such automation (for the purpose of this research) will usually involve computers.

The humanist approach, in contrast to Taylorism, is very prevalent in the research environment. The use of “research groups” where people are accepted into a community is a common example. There is a reliance on informal social interaction. Taking the Lancaster University Computing Department as a case study, in the departmental move to the new InfoLab21 building, emphasis was placed on breakout areas between offices as a key stimulant for synergy. The human-element has had such a large impact that the model’s impact is reduced to a few token gestures. A
process paradigm for research must take account of human factors both in terms of the individual researcher, and in terms of their environment.

The TQM paradigm seems completely opposed to the needs of the research environment. In computer science academic research the goal is typically to get a research paper completed and accepted to a conference. To do this “you need a flaky prototype that produces just enough statistics to give you tables and graphs to put in the paper. Once that's done …you want to move on to the next flaky prototype and the next paper” (Alison 2002). This approach does not rank quality very highly and sees any overhead at all as a significant challenge to be avoided. From TQM we learn that an over-concentration on quality leads to significant burdens and ultimately to rejection of improvement in favour of the status quo. Given the nature of the research environment, the tolerance for overhead is likely to be very low.

From these previous paradigms we can therefore describe the approach we need as:

- Automating repetitive tasks to improve efficiency
- Needing a very low overhead for any quality improvement approaches
- Taking account and being aware of the impact of human factors

These key points are discussed further as we examine the concept of automation in more depth in the next section.

### 4.2 Uses and limits of Automation

Historically, the idea of automating deterministic processes is what gave rise to the computer. The Jacquard loom shows one person plus suitable automation doing the job of an entire team. Babbage used automation to increase efficiency and reduce errors. Both Colossus and ENIAC demonstrate how through automation tasks that were previously too large to manage can be tackled. These benefits of automation through tools are added to the general idea of effort reduction from Taylorism as previously discussed. These are all benefits we can try bring to the research environment through suitable use of automation and other forms of process improvement. The limitation on deterministic processes is however critical.
Automation in the research environment will inevitably include deterministic processes being programmed into reusable tools. The scope for such improvement is real, but limited as research (and software development in general) is a work of continual improvement – not merely exact repetition. One area of automation has been generating documentation from source code. A number of tools undertake this task, the most well known is JavaDoc, the software that generates Sun’s API for Java. A similar tool to JavaDoc is dOxygen (van Heesch 2007), which has the advantage of working for more languages than JavaDoc and producing more types of output – including graphical diagrams when the right packages are installed. In this research we’ll use the example and adoption of dOxygen for the research environment. For real benefit auto-documentation tools require well-commented code as a starting point and for the user to treat the output not as a completed product, but as a starting point for their own work. In dOxygen’s case for example, small selections of the output (e.g. diagrams, HTML formatted code etc) can be inserted into reports and papers.

With the additional human effort, automation of deterministic processes allows an individual to do a team’s worth of work. It ensures accuracy of the documentation in a way that even a team cannot guarantee. It greatly improves efficiency compared to manual documentation, but also reduces overheads (another key point for a new paradigm). The effort to update documentation is reduced to a few clicks and a short wait. While the inclusion of an API to software is “interesting” it is only with the inclusion of relevant parts to illustrate a point in the report that it becomes “useful”, this is the human factor and steps to ensure the right ideas are captured in the document are a matter of software engineering rather than automation.

### 4.3 Making Software Engineering Fit for Purpose

The process of software creation has certainly improved in the last decades, however, the need to improve both programming and engineering is still with us. There has been very little work looking at such improvements in terms of their use to the academic computer science community. Royce’s two step process seems specifically targeted at the average computer science research process. It is in fact (consciously or not) what most researchers seem to use. Software engineering has been taught in universities without being applied by either students or academics. Common SDLCs such as the full waterfall and spiral models are too complex and rigid for research
projects. Research projects do not have the same kind of requirements that a project with an external customer would need. If we wish to minimize researchers’ overhead a new solution is needed. One such solution could be a greater focus on reuse.

Reuse remains a very important way to minimize effort, and if effort can be clearly reduced with a minimum of overhead costs, adoption is more likely. This is particularly true given the nature of the academic research environment where fast results are rewarded. While the current sort of software engineering may be too much of a burden (if applied as per industry) software engineering of some type must form part of the answer. After all, software engineering is the process of improving software development.

In further defining reuse, Cook, Ji and Harrison’s suggestion that repeatability might be less important than evolvability is critical for the research environment. Their four steps to software evolvability are included in the RAISER / RESET SDLC for the research environment (pg 4).

Combining the idea of evolvability with the suggestion that reuse is aided through customised frameworks (Oboler 2002), gives an approach to research involving bigger, longer term projects. In the RAISER / RESET approach, such projects would be maintained by one group of professionals, while a group of researchers reuse and adapt them as needed. The SERE project focuses on the role of the researchers in the first stage of such a development project and the software engineering tools and process improvement approaches they might use. In addition to being fit for purpose, the approach also need to be used. I.e. they need to be transferred to and between researchers. The next section discusses knowledge transfer in academia, highlighting some potential problems.

4.4 Technology supported process improvement
A technology based approach to process improvement is most common with either the organisational approach (where the organisation uses large scale information systems) or within the software engineering field (where technology is related to the core activity). Our approach needs to draw additional lessons from both these areas at both
Chapter 4  Discussion and development of key themes

the system level and the operational level. This section discusses these lessons and introduced a CMM style framework for describing departmental research capability.

4.4.1 Organisational Approach

The idea of an organisation approach has already been introduced in the introduction through the Research Process Optimization Framework, the RAISER / RESET SDLC and the concept of a Software Development Lab to assist researchers. These are however all novel approaches and in general academic research is focused on the individual. While we can draw lessons from the organisational approaches, the overheads involved make adopting any of them far from ideal in the research setting.

In industry in Europe, software organisations have shown support for Software Process Improvement. Focus has been at the organisational level rather than at the level of the individual, adoption at the individual level has been much lower (O'Connor and Coleman 2002). In the research environment this will be accentuated even more. There is a direct conflict between spending time on process improvement and spending time on the core business of research. Only when process improvement (for the current custom process) leads to immediate benefits that greatly outweigh the improvement cost can there be any hope of adoption in the research environment.

Business process review provides a perfect counter example to the needs of the research environment. BPR relies on “best practice”. It gives a high initial cost for changing the organisation, after which the use of more efficient processes is supposed to deliver benefits. BPR projects estimate the time till Return On Investment (ROI), that is until the benefits have accumulated enough to outweigh the initial cost. Given the variation between research projects, and the different languages, tools, and approaches to research used by researchers, it is unlikely there is a complete set of “best practices” that could be enforced. Without any commonality across a large range of projects, a return on the initial investment is unlikely in all practices are changed to that of a “best practice” model, as occurs during a BPR.

The example of BPR may however have benefits for research during the re-engineering stage. Within the RESET phase of the RAISER / RESET SDLC it would be possible to develop best practices. The development activity that occurs after the
research phase is much closer to a repetitive task. Though dependent on varying input, RESET tries to bring all projects to a common level, or a specific stage in a range of levels. It would aim to standardize all forms of documentation related to the project, from internal comments to API guides and user manuals. As the RESET Lab would involve a number of people and documented procedures, if a radical change were needed to the way a RESET Lab worked, a complete BPR might be appropriate.

Another set of organisational approaches are the ISO standards. These (like BPR) carry too much overhead for research projects (though some exceptions for very large projects may exist). Organisational process improvement (like ISO standards) generally have a “top-down approach, that gives little support to individuals and risks not addressing their day to day problems” (O'Connor and Coleman 2002). The ISO 9000 standards do not specify processes; rather they list business aspects and require that some documented process (which is defined by the organisation applying for certification) exists. Though far too heavy as it stands, there is some benefit to the idea of a list of requirements that the user then defines.

A light weight version of a list of aspects that need to be defined with in a process improvement model could be of use in academia, provided the overhead is small and the research is left in control of defining the actual process they intend to use. The benefit would be in helping researcher clarify their ideas. With a list of the various stages and an indication of possible approaches and tools, the researcher could check they weren’t over-looking shortcuts. The motivation would have to be of benefit to the researcher rather than certification for outside bodies or departments. An enhancement approach rather than an assurance one is needed. A RESET Lab is in a different situation and for a large lab certification either through ISO or CMM could conceivably have some benefit.

From the existing approaches discussed so far we can see that:

- Improvements must lead to immediate benefits that outweigh the costs
- There is unlikely to be a common set of “best practices” for all situations
- Researchers should be left to define their own processes
• An abstract list of aspects that could or should be defined can help researchers clarify their ideas.

The CMM approach, at least in broad terms, can be more directly applied than BPR, and ISO9000. The next section introduces CARE, a framework for the Capability (maturity) of an Academic Research Environment.

4.4.2 CMM and the CARE Framework

University Research environments are usually at the initial stage of the CMM. Projects work or fail based on the efforts of the individuals. Beyond project proposals and financial management there is very little organisational control of an individual project. Within a project, the approach is usually ad hoc.

While not the intended use, we contend that the idea behind CMM and the concept of maturity can be of benefit in the research environment if applied not to the organisation, but to the specific project.

Capability of Academic Research Environment (CARE)

In a maturity structure based around an individual research project, the project could move beyond the initial phase if an approach to the research, (including the use of guidelines, tools, libraries and reusable code) is planned out in advance. This is not suggesting the project be strictly designed and documented, but rather that the process be planned and documented. A similar project would significantly benefit from this planning and as a result the process would be repeatable (i.e. stage 2 CMM). Instead of cost, schedule and functionality as key factors of project management, I’d suggest that for research the primary metrics to understand would be the usefulness of various tools to this sort of project, the effort required to learn tools, the effort and time required to understand inherited code, and the effort required to adapt inherited code. This is assuming that a similar project means one where the same tools and code can be reused, or at least one in which the same tools and code would need to be considered. This follows a shift from repeatability to adaptability as recommended by Cook, Ji and Harrison (Cook, Ji et al. 2000). We find this shift much more useful and relevant to research.
The CMM level 3 “defined” is harder to adapt to research. The idea defining a development process so it can be reused is particularly un-useful for the research environment. The process of development will vary so radically between projects that similarities are likely to exceptions rather than the rule. Customisation would therefore typically involve throwing out the standard and starting again. This is not useful. What we feel can be defined is the process of creating a customised research process. In other words, documented guidelines that describe the things to consider in formulating a custom research process. For some projects, a very similar project will already have had its process documents, so the first step of such a process could be to check if there is an existing template to alter. If such templates can be stored and managed effectively, with new process plans added when they are created, the research environment could be said to be “defined”. This would be one of the goals of a RESET Lab.

The CMM level 4 “managed” criteria could be adapted as a result of the level 3 adaptations. Data on the effectiveness of the process can be added and the reused code improved to make it easier to use next time. This would introduce the RESET process and the start of RAISER/RESET cycles.

The CMM level 5, “optimising” stage, could be achieved in the research environment with a RESET Lab which actively tries new ways of improving the research process. This could range from changing the style of documentation they use to the creation of interfaces to software that is used regularly.

As discussed above in regards to ISO 9000, either CMM or ISO 9000 certification could be beneficial for a large RESET Lab.

From the CMM approach we have derived:

- A framework for discussing the relative maturity of research environments
- A set of goals for process improvement when considered at a systems level

### 4.4.3 Existing Personal Approaches

While personal approaches should be more suited to the research environment than the top-down organisational approaches, they are still falling short of the needs of...
researchers. The cost of adoption and use still appears to outweigh the benefits in all cases.

“Given that PSP demands between 100 and 200 hours of initial formal training and an extremely high burden of data collection and analysis, it is unlikely that small companies with limited time and financial resources would risk starting their process improvement program with the PSP” (Ward, Fayad et al. 2001). Likewise, starting with researchers is probably far too late for such an intense training process. An intuitive feel of progress and short-term benefit may be far more productive as an incentive than the PSP’s reliance on empirical data.

Based on (Humphrey 2000), O’Connor and Coleman (O’Connor and Coleman 2002) in their introduction of PIPSI conclude that “using a defined, planned, and measured personal process appears to offer many benefits”. While suggestion that both PSP and PIPSI could provide a framework to offer these benefits, they suggest PIPSI, being more lightweight, provides a solution that is more likely to be adopted for SMEs. For the research environment both PSP and PIPSI are too heavy. Using a “defined, planned, and measured personal process” is however a sound recommendation. For research, “measured” should mean “limited” and “personal”, i.e. not just for individual use, but also tailored to individual needs. While industry approaches may have too high a cost for the research environment, low cost approaches developed for the academic environment may have a warm reception in industry.

While PIPSI has aimed to reduce the burden of adoption (down from 2 week for PSP to a mere 2 to 3 days (O’Connor and Coleman 2002)) this is still far too much. A course can teach tools, and exercises can give students an experience of thinking in new ways, but unless this leads to a significant time saving for students it is unlikely to provide more than academic value in the university setting. To achieve a time saving in the research environment, a personally tailored approach must be developed for each type of project undertaken.

PSP/Baseline is the best solution examined, with its focus on an individualized development process and separation of “Instantaneous benefits” from “Continuous benefits” (Johnson 1997). These are points we strongly support. The lack of adoption
of even this approach is of concern given it takes place in an academic environment and meets most of our suggestions while avoiding many of the aspects that make other frameworks inappropriate. Examining the barriers suggests that the collection and use of statistical data is again a problem (as with PSP and PIPSI) the process improvement is seen as an additional step separate from main project, and often of little benefit to the student. The conclusion for research is to move away from an intellectual statistically based approach to one of instant gratification – focusing almost entirely on the “instantaneous” benefits and only promoting continuous benefits when they require little effort upfront, but will a priori deliver a valuable benefit later. Adding documentation of information clearly needed for a report or paper would be one example of appropriate continuous benefits.

The following can be added to our lessons:

- The learning cost is a significant barrier to adoption
- An approach should be limited and personal, i.e. tailored to individual needs
- The same researcher may require different approaches for different projects
- Benefits should be separated into instantaneous benefits and continuous benefits
- Most of the focus in academia should be on instantaneous benefits
- Continuous benefits should be carefully justified
- Process improvement should be integrated and not an additional separate task

4.5 The Process Framework

The greatest lesson from Osterweil’s landmark paper (Osterweil 1987) is that of the “doubly indirect” nature of process improvement. He explains in some depth the different layers of process abstraction and process improvement. The framework seems reasonable for the research environment and is consistent with, and helps further develop, the idea of RAISER / RESET and the Software Development Lab.

The goal of process programming is “the creation of a process description which guides the specification and development of an object which in turn guides the specification and development of another object which solves problems for end users” (Osterweil 1987). For clarity, another object would refer to research output. An object would refer to an individualised process descriptor that outlines how to produce that
research output. The initial *process descriptor* refers to the set of steps, tools and approaches used to guide a researcher in creating their own process descriptor object. The concept of a process descriptor is explored in more depth in section 4.6.

The use of process descriptors within the research context will be elaborated and extended in this thesis, with the aid of diagrams below. The diagrams are inspired by those in the original paper, but adapted by us to UML notation and to the context of the research environment. This material introduces the concept while keeping within the context of this research to aid understanding.

**Level 1: The research process**

![Figure 6: The research process](image)

In Figure 6, a research creates a proof-of-concept which includes (for example) a proposal, literature review of similar work, identification of the research gap, a software architecture, software, evaluation methodology, results, conclusions and list of areas for further work.
Level 2: The research process as an instantiation of a Process Description

Figure 7: The research process as an instantiation of a Process Descriptor

In Figure 7 the research process is shown to be an instantiation of a process description for making “research”. This abstract descriptor is created by a software engineer who improves the abstract model based on feedback.
Level 3: The Process Improvement Process

In Figure 8, the researcher carries out the process of process improvement. They do this by becoming critically aware of their actions and choices (their process). They are aided by having Process Descriptors available. A descriptor that is derived from the basic template for “making research” but is concrete and tailored to their field (e.g. computer science or a specific area of computer science) provides greater guidance in improving the process. The researcher still needs to add their own process customisation, and instantiate various parts of the process descriptor that are left as abstract.

The Process Descriptor Template differs from a CMM level 3 standard development process by acting as an abstract template rather than as a master plan. It is not the plan for the research but rather a plan for creating that plan. In this sense it has a lot in common with the CMM or ISO descriptions, but is tailored to ask for the right things for this type of project.
Level 4: Process of Improving Process Improvement Processes

Figure 9: Process of Improving Process Improvement Processes

Figure 9 shows that to continually improve the process of process improvement, the Process Descriptors must themselves be added to, updated and reviewed. This includes updating the template as well as adding new and varies process descriptors based on the experience of past researchers. The role of the Software Development Lab and the cyclic nature of RAISER / RESET would make this review possible. Continual improvement also involves evaluation of new process descriptors (including software tools) and the gathering of data on what researchers actually do. The current research forms part of this process.

The level we are working at is the top conceptual level. Figure 8 shows the realisation of a process, Figure 9 shows the application of engineering to these processes. It must be stressed that the engineering practices used are the same as in the research process itself. To uses an analogy, the programming of a programming language is no different (in approach) from the use of that language to programming something else.
Chapter 4  Discussion and development of key themes

The tool-set we make available to others is the same as the tool-set we use for improving our own approach, and tools we (as the software engineer) find useful for doing this research should also be evaluated for use by the other researchers we are assisting – this is based on Osterweil’s observations, which to us appear fundamentally sound if properly applied.

The process development process (creation of new process descriptions) and process evolution process (improving a process description based on experience) exist at the top (most abstract) level in the hierarchy. To improve these processes themselves is the same as improving any other process description. It is (to continue the analogy) using a programming language to make new extensions to the language.

4.6 Process Descriptors

In section 4.5 we introduced process descriptors and gave them a formal definition. In this section we explore this idea further and place it in the context of our research.

At its broadest level, a process description is simply “a specification of how the job is to be done” (Osterweil 1987). They are developed to meet the requirements of the process and take account of its limitations. In this research, looking at process improvement in computer science research in academia, our process descriptions are artefacts that automate or guide the researcher through parts of the research process. Each researcher should also have a process descriptor for their overall research process. We will facilitate process improvement by making researchers aware of their own personal research process and encouraging them to take control of this process and engineer it to meets their satisfaction – not simply allow it to happen to them.

Osterweil notes that, “it should be clear that there is no ‘ideal software process description’… We believe that expert software managers have intuitively understood this for quite some time and have become adept at tailoring their processes to their needs” (Osterweil 1987). In the case of research, each researcher is their own manager and as such needs to tailor their own process descriptor for the over-all research process as well as customizing any descriptors given to them for their own use. Because of this we will present template descriptors in this project mostly as outline suggestions or examples of varying applications of tools we make available.
Osterweil believed “that a software environment is best viewed as a vehicle for the specification of process programs, and for their compilation and interpretation.” While wanting to move away from the idea of coding processes, we do see one area where this idea can work well. For the over-all process a customized UML class diagram would seem appropriate, provided researchers do not need to manually create it. The creation of a tool would remove the burden others have seen as such an inhibitor to the adoption of process improvement. A tool could generate code stubs as an intermediary step and then use dOxygen comments to record details about each step in the process. Finally it could generate class diagrams and an API as the final output of the “process programming” activity. The user can edit the “source” code to add further comments and develop their process while familiarising themselves with dOxygen at the same time. This is very close to the original process programming idea, but removes many of problems and shifts the focus to being purely high-level process design. In summary it is a combination of:

- The Process Programming idea
- A modern approach to presenting programming ideas (UML)
- A restriction to high level ideas (Class diagram)
- Incorporation of free text to document ideas and aims
- Generation of an API that can be used to navigate aspects of the project’s process as more detail is added

4.7 Use and Reuse of process descriptors

In order for process descriptors to be reused effectively, a process to both archive and find useful descriptors is needed. The process to find and evaluate other descriptors and software for reuse should occur “only at the proper points in the larger process of developing and reusing software” (Osterweil 1987). While a RESET Lab would stabilize software and make it ready for reuse, (a proper time for this, outside of the usual research time frame), within this project it is also important to introduce process descriptors at the right time. To aid this in an environment when different participants are progressing at different speeds, process descriptors can be available on the web and reminders of what is available can be e-mailed when the time appears to be right. A further idea of tutorials and regular group meetings is being considered, these again
would allow process descriptors to be considered at the proper time for individual researchers.

There is a question about the reusability of process descriptors. Osterweil expects “that early process programs will be produced from scratch by software engineering researchers, but that in the future process programs will be customized by working engineers out of standard process programs” (Osterweil 1987).

For the research environment specifically, the reuse of descriptors of the overall research process does not seem likely, except where the research involves adapting or rewriting an existing program. This said, we reiterate our hope that a basic template can be made available for researchers, from which they can build their own research process description and document process they create that are found to be helpful. Self-contained process descriptions may well be reused if they capture useful ways of working or using tools to greater effect. There is a danger of trying to produce a “standard approach” that is then only mildly altered for different people and projects. In the research environment at least this would not be ideal as it would inhibit freedom and new ideas and approaches to an unacceptable degree. It is akin to providing a vanilla ERP package and making all projects customise it slightly to meet their unique needs, in the research environment projects are likely to have more differences than commonality.

The most useful approach is likely to involve making tools and other descriptors for specific tasks available, along with a way of organising these and recommendations for their use so they can more easily be incorporated into a cohesive individualized plan for a researcher. This is why the Process Descriptor Template is left as a template rather than developed into a CMM level 3 style standard development process. It is also why the process descriptor should be reviewed and changed as the project develops. “The processes by which software is developed are likely to change with circumstance — perhaps even change dramatically—even while general principles like the need for good communication remain constant” (Ward, Fayad et al. 2001). This is even more pronounced in the research environment where “any SE approach for research software (that would have any chance of adoption) would have to be agile and evolutionary in nature” (Pressman 2002). As a result, the Process Descriptor
Template needs to include general principles while not specifying their implementation. An open set of examples could be useful, provided it is made clear that other approaches developed by the researcher may be better for them.

4.8 Goal and Risk Driven Processes

Ward, Fayad and Laitinen (2001) suggest that “consideration of appropriate process be goal and risk driven”. They illustrate this with the example of a prototype development comparison to the release of a version 3 product noting that the two “will have a different set of goals and risks”. Research is by definition working at the prototype level. The goals of the researcher and the risks they face form the basis for the development of our approach to process improvement.

(Ward, Fayad et al. 2001) also gives us some fundamentals explaining that, “a process is a tool rather than an end in itself… improvement comes from management assuring that tools are properly applied.” In the research context, the researcher is their-own manager. An exception could be made with MSc projects, by setting deadlines for various stages and attaching marks to these submissions… but this was not the situation during these case studies. A corroborating statement would be that the process description (like any design) must be regularly checked and updated… without the researcher’s attention it does nothing useful, and perhaps even some harm as benefits are assumed.

Another recommendation is that, “processes must be simple… complex processes are difficult to follow, difficult to update, and quickly become unsuitable for the operations they originally specified” (Ward, Fayad et al. 2001). Particularly for the research process, the process descriptions must of necessity be lightweight. A software engineering burden will prevent research from occurring.

Ward, Fayad etc al. (2001) note that “while the cost of quality may be free in the long run, a small company, especially a startup, cannot ignore its up-front costs.” The same is true for research. There is no return on effort invested in getting it perfect (Oboler 2002). Ward, Fayad etc al. (2001) also say that “common sense dictates that processes be established that match the needs of the organization and grow with it rather than being imposed by some abstract ideal.” Finally, “Processes must be
robust… [i.e.] easy to apply and… difficult to get wrong without warning indications” (Ward, Fayad et al. 2001).

4.9 Summary of key concepts
The lessons drawn out of this chapter can be divided into nine key themes. These are presented below as a summary of ideas, followed by some concluding remarks to the chapter as a whole.

Conceptual framework
1. A process should be considered as a set of steps, with related artifacts, tools, resources, organizational structures and environmental, personal and subject based constraints.
2. Process improvement must not be seen distinct from the main project, or it will appear as a distraction of little benefit to the student. A focus primarily on instantaneous benefits is one way to avoid this separation.
3. The process framework with its four levels conceptualises the notion of the process of process improvement. It provides an overall view of the process, the role of the actors and the use of process templates and descriptors.
4. This research can be described as the Process of Improving Process Improvement Processes, i.e. it is a process of meta level research
5. The process of process improvement must be carried out by the researcher. They do this by becoming critically aware of their actions and choices (their process). They are aided by having Process Descriptors available, which they can use, as is, alter or create alternatives to.
6. The engineering practices and process descriptors presented as part of this research are also used (to varying degrees) in conducting this research. Personal experience of their use as well as feedback from participants can therefore lead to changes and revision.

Approaches to improvement
7. Effort can be reduced through process automation for exactly repetitive tasks.
8. Automation can allow researchers to do the work of more people, increase accuracy, and tackle larger tasks.
9. Deterministic processes can be programmed into reusable tools. The use of such tools should be a starting point to using their output and customizing it.

10. A limited, but defined approach that is personally tailored to the needs of an individual researcher seems the best way forward.

11. Through objectifying the development process it becomes easier to understand it and as a result easier to analyse and systematically improve it.

12. Process descriptors need to be considered at the right time. To do this they should be available on the web and reminders of what is available should be e-mailed when the time appears right. If tutorials and regular group meetings occur, these too would allow the timing to be fine tuned and personalised.

**Guidelines for tools and techniques**

13. *Reuse* is the best way to minimize effort, and repeatability is less important than *evolvability*.

14. A process is a tool, not an end in itself.

15. Processes must be simple.

16. Diagrams produced by hand should be restricted to a high level (e.g. Class diagram)

17. Processes must be robust, if used wrongly there must be warnings.

18. Process descriptions are artefacts that *automate or guidance the researcher* through parts of the research process, they give a *specification of how a job is to be done*

19. High-level process descriptors are *guidelines to a general approach* UML class diagrams may be suitable for the design of both architecture and process

**Specific tools supported by the literature**

20. Incorporation of free text to document ideas and aims

21. An abstract list of aspects that could or should be defined can help researchers clarify their ideas.

22. *A tool that generates documentation of their process* as a UML diagram / API documentation should be possible. The use of such a tool might raise awareness of the process and encourage planning.

23. Each researcher should have a *process descriptor for their overall research process*. This needs to be built by them and tailored to their project and style.
Chapter 4  

Discussion and development of key themes

Awareness and planning of the process is a critical step towards taking control and improving their own process.

Reuse

24. Reuse (of suitable code) is the best approach to improve efficiency, it does this by moving the starting position (in Research Process Optimization Framework terms).

25. Self contained process descriptions may have reuse potential if they capture useful ways of working or integrating tools.

26. The reuse of descriptors of the overall research process does not seem likely, except for projects of rewriting or adaptation. Process descriptors that form part of the overall model can and should be shared.

27. It is unlikely there is a set of “best practices” that could be productively enforced in the research environment.

Adoption

28. Researchers should be encouraged to define their own processes.

29. An over concentration on quality leads to significant burdens and ultimately to rejection of improvement in favour of the status quo.

30. New approaches will need to be taught in order to reduce the overhead in learning.

31. In existing process improvement methods there is a direct conflict between time spent on process improvement and time spent on core business. Improvement effort must lead to obvious and immediate real world benefits.

32. While a course to train new students is a valuable approach, it must give significant time saving to be of long-term value.

33. There needs to be a clear separation of “Instantaneous benefits” from “Continuous benefits”. Only intuitively obvious continuous benefits are likely to be adopted.

Environment

34. Humanist approach is mainstream within the research environment, human factors have to be strongly considered as a major impact on research.

35. SDLCs such as the waterfall and spiral have too much overhead for the research environment.
36. The cost of adoption of personal approaches (e.g. PSP, PIPSI, PSP/Baseline etc) still appears to outweigh the benefits in all cases. *An intuitive feel of progress and short term benefit may be more productive than reliance on empirical data.*

37. A standardized approach is not a useful approach in the research environment.

**Systems view of improvement with CARE**

38. Our CARE framework introduces a mapping between the outline of CMM and the research environment – concentrating at the level of an individual project. University research environments are usually at the initial stage of the CMM.

39. To move to the second (repeatable) phase, *usefulness of tools*, the effort required to *learn tools*, the effort and time required to *understand inherited code* need to be understood.

40. To move to the third level (defined), the tools etc must be *incorporated into templates* and used by researchers to *build process descriptors* that outline their research process. These also need to be stored and managed effectively.

41. To move to the managed level of CARE (level 4), data on *process effectiveness* need to be added and projects need to be *improved so it is easier to reuse them* (i.e. they need to be RESET).

42. To move to an optimising (level 5) stage the approach and tools need to be *experimented with to get best results*.

**Lessons for the Reset Lab**

43. Best practice could be employed in the Reset Lab.

44. A large RESET Lab may want to be ISO 9000 or CMM certified.

45. To achieve a defined level in CARE, a Reset Lab would be useful to manage the templates. To achieve a managed level a Reset Lab is required to reset projects before they are reused. An optimised approach would be best achieved through a RESET Lab which systematically tries new tools and approaches to improve the research process.

46. The process framework specifies the role of the Reset Lab in the research environment.

47. The Reset Lab has responsibility for adding to, updating and reviewing Process Descriptors.
48. The Reset lab can ensure process descriptors are considered at the right time and the right information is made available. This would allow both process descriptors and code to be reused.

4.9.1 Concluding Remarks
Many have looked for the one invention that will dramatically increase productivity in software development. In 1986 Fred Brooks explained that “There is no silver Bullet” (Brooks 1986), meaning that there is not, and will not ever be, one single tool or paradigm that will deliver a tenfold increase in productivity. “The complexity of software is an essential property, not an accidental one.” He explains that software is pure “thought stuff”, abstract away its complexity of concept and you wish away the software itself (Brooks 1986). This is true of research too. Researchers are by definition working at the leading edge. Brooks notes that the most valuable part of software development, is often not the part that takes the most time. “With any creative activity comes dreary hours of tedious, painstaking labour, and programming is no exception” (Brooks, 1995). In our previous work we rejected the idea that “research is pure inspiration and therefore the research process cannot be improved” (Oboler 2002). This left us with the question of how research could be improved and at what cost.

As we have seen, even Taylor only managed a four-fold increase at best with process improvement. Process improvement is therefore unlikely to deliver a silver bullet effect. It does however provide a path to improvement and in our context specifically, it aims to reduce the “dreary hours of tedious, painstaking labour” so researchers can focus on the essence of new research. As the research takes place in-vivo the subjects themselves will control the cost. If they do not feel they are getting benefit, adoption will not occur. This leaves the question of how the research process can be improved – the topic of this research.
5 The SERE approach: Overview

In this chapter we introduce the SERE approach, as it stands after the conclusion of our research. An overview is presented, followed by an introduction to the various components within SERE. The SERE approach was developed in an evolutionary manner, the details of which are presented in Chapter 6. The evaluation of the components introduced here can be found in Chapter 8 and an evaluation of the overall impact of SERE can be seen in Chapter 9.

5.1 Introduction to the SERE approach

The SERE approach aims to improve the research process in academic computer science research through the provision of tools, guidelines and interventions which we generically refer to as process descriptors. These process descriptors are used in a context where academics manage their own research, but have access to in house software engineering expertise (a software development lab) to advise on process descriptors, arrange reviews and eventually reset the final research product to make it easier for others to build upon it in the future.

An improved research process could include benefits such as increased knowledge retention, improved collaboration, increased product quality, better quality research, improved satisfaction with the process and / or product, a more systematic approach, and improved efficiency. Although not a benefit itself, adoption of the approach is an important indicator of improvement and without adoption the other benefits remain purely theoretical.

To ensure adoption, the SERE approach aims to provide maximum immediate benefits at very low cost. Longer term benefits, and particularly benefits for future researchers are linked to immediate benefits for the current researcher. The impact on the researcher’s approach to their work is kept small both in terms of time commitment and in terms of changes to the way they work.

The overall meta-approach specifying the role and interaction between researchers, the software development lab, and the process descriptors has already been presented in section 4.5 (pg 66) and forms one component of SERE. Other components are designed for use by the researcher, and although SERE is presented as a complete
approach, each of these components could be used or rejected as the researcher deems appropriate. SERE also includes one process descriptor designed specifically for the software development lab, a re-engineering pattern known as “read everything in two hours”. These components will be introduced in the remainder of this section.

5.2 Introducing the components of SERE
In this section we introduce some of the key components of SERE. The term components is used to include process descriptors, meetings and other aspects that can not rightly be considered process descriptors as they are events in the process and not artefacts. An overview of the components, divided into categories, is shown in Figure 10.

![Figure 10 Components of SERE](image-url)
Components examined in this chapter include the Process Model for Supporting Research, the Personal Process Model, the use of High Level Diagrams, Documentation and dOxygen, Coding Guidelines, Formal Technical Reviews and the use of a reengineering pattern to facilitate the engineer. These items are presented in detail here due to their large impact, generic nature or their new and novel contribution. In all cases there is supporting discussion on their use which is provided in Chapter 8.

5.2.1 The Process Model for Supporting Research

In Figure 11 the Process Model for Supporting Research is presented. It shows a software engineering team (or Lab) creating process descriptors, assisting researchers who are creating process descriptors, and maintaining the catalogue of process descriptors so they can be easily shared with the right tool offered for the job. In addition to feedback on the descriptors in the library, the Software Engineering Team would also receive requests for new process descriptors to meet specific needs.

Figure 11 shows how multiple templates can be used as starting points. Ideally there would be a template for each research group / area that lists the tools, procedures and people they should contact. This would help new members settle in fast and more rapidly acquire the knowledge that is often not transmitted explicitly. The researcher
on the right of the diagram can be seen to be choosing a template to start from. Researchers may also create new, more specific templates for their group, the create action is therefore also shown, though this would be a more rare occurrence.

The researcher (being critically reflective) selects process descriptors and adapts them so they become more suitable for their purposes. This is shown with the inheritance arrow. Even with more specific process plans for individual research groups, each person still needs to adapt their overall process and their own use of any tools or guidelines. The researcher does this in consultation with their peers. All of this takes place in a meta-context that is outside the scope of the researcher actually doing research.

### 5.2.2 The Personal Process Model

The Personal Process Model is a new way of organising, discussing, and self-managing the research process. As discussed in section 5.2.1, the personal process model allows researchers to take control of their own process by modelling it in a flexible and dynamic way, using source code and code comments (a template is provided to get started). The dOxygen software is then used to generate a report.

The “generic template” has been provided in the appendix (pg 273) along with a sample page of its generated output. The report includes UML style diagrams of the process. The output (like the dOxygen output for the code) includes more detail in the comments that hide behind the method and property names shown in the diagram. The classes in the diagram may be clicked and serve as a navigation aid. The documents convert linear code into true hypermedia that allows researchers to focus on a particular area while still being aware of the context of the entire research process.

The personal process model is a new idea adapted from Osterwel’s idea that “software processes are software too” (Osterweil 1987) and combining it with the benefits of generated documentation and object orientation. This combination leverages existing skills of computer science research to help them manage their research process.

### 5.2.3 High Level Diagrams

The concept of “high level diagrams” is used to denote architecture diagrams, component models and similar (possibly abstract) divisions of the research product into smaller manageable modules. While too much engineering can be inefficient in
the research process, some level of planning is worthwhile and indeed required to process beyond the initial stage in the CARE framework. The efficient researcher needs to locate and isolate the interesting parts, separating the experimental work from the mundane coding tasks. These experimental tasks then need to be given a higher level of attention, while other parts of the code may receive little planning beyond the outlines.

Diagrams do not need to be electronic, though this can make it easier to share them with others. For SERE, the warning not to produce detailed design documents (wasting time) is as critical as the need for a high level diagram. The form of the diagram and the software used to create it is not important, though they should not require a large amount of time to do the job, and should ideally allow changes to be easily made if needed.

5.2.4 dOxygen and Documentation
Documentation is a key component of the SERE approach as it is through documentation that researchers can reflect, gain insight, and share their ideas. We aimed, however, for a “light touch” approach and ways of integrating documentation with the research process in a way that does not cause researchers to ‘task switch’. As a result almost all of our documentation recommendations integrate with dOxygen, a tool that extracts comments from source code and produces a report based on the comments and code structure. Consistent use would allow all the documentation for a project to be generated as one set of hypermedia.

In addition to the dOxygen software (open source), SERE includes an installation and setup guide for dOxygen (appendix, pg 263). The SERE Coding Guidelines (pg 85) include templates for coding with dOxygen and the personal process model (pg 82) is created through a template and dOxygen. The dOxygen output features in our reengineering pattern (pg 87) that was used to prepare for formal technical reviews, and in the reviews themselves (pg 85) which involved a walk-through the dOxygen output. Sample code and dOxygen output from a past participant were also provided online, allowing researchers to judge the cost and benefit of using dOxygen.
In any SERE approach it would be necessary to provide a similar level of assistance in setting up and using software, as well as finding means of allowing people to make a realistic judgement about the costs and the benefits to themselves of using a certain process descriptor.

5.2.5 Coding Guidelines
The Coding Guidelines focus on documentation and touch upon coding style (appendix, pg 251). Unlike regular coding style guides, the SERE Coding Guidelines are designed to capture information about the research process and why decisions have been made. Equally important is information about the level of effort that has been invested and ideas that have been tried and abandoned. The coding guidelines have been published separately (Oboler and Sommerville 2007).

The guidelines aim to move researchers away from documenting what code did, and on to a more mature level of documentation where a reader’s understanding of the basic operation of code can be assumed. Many students, realising that their comments added nothing to the code, stop commenting as they progress and become researchers.

The Coding Guidelines encourage researchers to document those things not present in the code itself, such as design rational or the reasoning behind the selection of particular algorithms. The guidelines also encouraged researchers to capture a certain amount of process information, specifically how much effort they had spent on parts of the code in certain circumstances. This information can be important, e.g. a particular search algorithm could be used because it was the first thing that occurred to a researcher, or it could be used after weeks of testing various algorithms. The implementation is the same, but the implications for someone wanting to adapt or improve on their code is significantly different. An algorithm that was selected randomly is a much clearer target for improvement than one where significant amount of effort has already been invested.

5.2.6 Technical Reviews
Technical reviews are both a specific approach used in SERE, and a well defined approach more generally. We begin with some background and our own past research before moving to a discussion of Technical Reviews in the SERE context.
Technical Reviews are the most widely used approach in Software Development (in Industry) to validate the quality of a product or process (Sommerville 2001). Despite this, a re-analysis of data we collected for the USE CSR project shows that they are seldom used in academia, and in fact many academics don’t know what technical reviews are. This is shown in Table 4.

<table>
<thead>
<tr>
<th></th>
<th>Don’t Know</th>
<th>Don’t Use</th>
<th>Use Rarely</th>
<th>Use Sometimes</th>
<th>Use Often</th>
<th>Use Always</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qty</td>
<td>15</td>
<td>11</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>%</td>
<td>43</td>
<td>31</td>
<td>3</td>
<td>17</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4 Use of FTRs by Australian Academics (Oboler 2002)

Johnson (Johnson 1999) explains that a technical review is “a method involving a structured encounter in which a group of technical personnel analyses an artefact according to a well-defined process.” In his book *Software Engineering*, Sommerville explains that the remit of the review team is “to detect errors and inconsistencies and point them out to the designer or document author” (Sommerville 2001). Johnson adds that the outcome of a technical review is “a structured artefact that assesses or improves the quality of the artefact as well as the quality of the method” (Johnson 1999). The outcome of a review is therefore not only constructive but concrete.

Sommerville extends the list of products that can be reviewed, noting they are “document based but are not limited to specifications, designs or code. Documents such as process models, test plans, configuration management procedures, process standards and user manuals may all be reviewed” (Sommerville 2001). Additionally, research protocols provide a domain specific document type that can also be reviewed.

SERE reviews take account of all these points.

In the SERE approach, researchers code, as well as the dOxygen generated documentation of both code and their Personal Process Models are reviewed. The reviews concentrates not just on the product but also on the process.

The nature of Technical reviews vary, depending on their purpose and position in the SDLC. Tjahjono (1994) describes Technical Reviews in terms of three orthogonal dimensions: Objective, Interaction Mode and Technique. The objective mode is usually described as one of comprehension (understanding the material under review),
examination (reviewing the artefact with a mind to finding errors) and consolidation (reviewing errors spotted by individual preparation or past reviews).

In SERE reviews the focus is on consolidation, at times simply looking for inconsistencies or “gaps” in the explanation provided by the researcher’s documentation. The interaction mode used in our study was synchronous, and the technique was semi-structured and at times free form. These were implementation details and other options could equally suit a SERE approach.

A technical review preparation document was provided as a process descriptor (appendix, pg 270) and gave guidelines on what researcher should have done before scheduling a review.

In order to carry out technical reviews, an additional further process descriptor, the Re-engineering Pattern “read everything in two hours” had to be created. This process descriptor (discussed next) was not for the students but for the Software Engineer. The main feature of the technical review from the RESET engineer’s perspective is the opportunity to discuss a research project with the researchers. This is the only real opportunity for the RESET engineer to highlight areas where documentation may not be sufficient to prevent the loss of valuable research information. It is also an opportunity for the engineer to encourage better structure and engineering by the researcher, making the engineer’s job more efficient and of higher quality in the longer term.

5.2.7 The Re-engineering Pattern
Through their projects, participants received similar assistance to what we would expect an in-house RESET lab could provide to researchers. As all the assistance (technical reviews and meetings) was provided by one person, and the number of projects was large and diverse, the engineering challenge to understand students’ work and be able to provide meaningful technical reviews was significant. We found two patterns from “The FAMOOS Object-Oriented Reengineering Handbook” (Bauer 1999) that were of help and after using these in the first year, and we merged and adapted them to form a new pattern that was more in tune with SERE. The new pattern, “Read Everything in Two Hours”, is presented in the appendix (pg 260), and was then used for the final two years.
The original patterns were “Read all the code in one hour” (Bauer 1999) and “Skim the Documentation in an Hour”. Both are designed for use at the start of a reverse engineering project on a large and unfamiliar software system. The first pattern warns to beware of code comments that may be out of date, and the second only applies to external documentation that is produced before or after the code. The new pattern for SERE is designed to cover both documentation rich code and auto-generated documentation based on that code. A particular feature of this situation is that much of the information will be duplicated and does not need reading twice.
6 Developing SERE: The Experiment

In this chapter we present our methodology to examine the various research questions examined in this work. Our primary question is whether a RAISER-based approach can improve the research process in computer science academia. The null hypothesis is that a new approach will make no difference.

At a more basic level, the question of whether software engineering (of any type) can be a suitable aid in the academic environment is an open one. While a positive answer to our primary question is sufficient to positively answer this underlying question as well, the null hypothesis here differs. The hypothesis that software engineering (of any form) is not appropriate to the research environment would lead to an expectation of very low adoption of our approach, or if adoption occurs, would lead to detrimental results and a negative over-all review of the experiment by participants. Our experiment is designed to capture information on this more basic question as well as on our recommended approach.

As the RAISER approach only provided an abstract outline, a specific approach with concrete artefacts needed to be developed, we called this SERE (Software Engineering in the Research Environment). Each component of this new approach would need to be checked for compliance with the RAISER principles, as would the approach as a whole. Exploratory research is also needed to develop the SERE components and confirmed the results through repeated trials.

To answer the questions above, we need a number of experimental cycles with improvements to the experimental design occurring between cycles. Within each experimental cycle we need a number of projects, some using the SERE approach and some using the default behaviour. We also require that enough data be collected to measure “success” both of the parts and of the whole.

This chapter presents our experimental choices, our design and details on the various data collection methods used.
6.1 Experimental Choices
There are a number of experimental choices that had a significant impact on the experiment. These choices were designed to increase the power of the experiment, giving us greater confidence in our results, while also ensuring the experiment was practical to run. Once of these choices (participant selection) was altered during the course of the experiment, though all the choices discussed here were intended to be consistent right through our experiment. These choices lay out our experimental foundation and differ from other higher-level experimental choices that evolved as the experiment progressed. After laying the common foundations in this section, the evolving factors will be introduced in the next section on experimental development.

6.1.1 Selection of Environment
We chose to perform our experiment in vivo, i.e. under normal conditions. The other alternative would be an in vitro study, i.e. one under controlled conditions, e.g. working on toy problems in class. The use of real projects gives a richer set of results, but allows other real world factors to have a significant effect on results. Such factors include the variance between the projects (interest, difficulty, risk) and the influence of the different supervisors. We mitigate these differences through our choices in the selection of research subjects, approach and metrics. We also make extensive use of interviews and open questions in collecting our data, turning these factors from a level of background noise in our quantitative data into a rich set of qualitative data on research under different conditions.

Basili (1996) states that experiments can be descriptive, correlational or cause and effect based. Our primary research question (whether a RAISER approach is of benefit in the research environment) requires a causal examination, but due to the complexity of the experiment this will involve a large amount of descriptive investigation as we learn more about researchers’ approaches to their work and refine our methodology approach.

While a causal relationship between certain aspects can be established (for example, the promotion of documentation guidelines and the existence of source code documentation), in other areas the relationship is impossible to determine beyond the level of a correlation. Due to the nature of this research we have also collected a large
amount of interesting descriptive information, much of which is tangential to our research question and therefore not included in this thesis directly. Such descriptive studies have however informed our approach and tools.

### 6.1.2 Selection of Research Subjects

MSc students working on five month research projects were chosen as the research subjects. Other options either did not provide a research project, or would not have been completed in sufficient time to allow results to be gathered and the experiment replicated. The experiment was designed with three cycles (each with a new cohort of MSc students) to allow refinement and incremental improvement of the SERE approach. This improvement as well as other research activity took place outside of the five months of MSc project time, when data was being collected.

The students were all given an introductory lecture and offered an opportunity to take part in the experiment “Software Engineering in the Research Environment” (SERE). The RAISER / RESET SDLC was presented and the nature of SERE as an implementation of RAISER was explained. At the lecture students were asked to indicate whether they were interested in participating. Closer to the start of their projects, all students were asked via e-mail if they wished to sign up. While there was a large amount of overlap, not all students who initially expressed interest signed up, and not all those who signed up had initially expressed interest.

We initially planned our approach as a *controlled experiment* (Basili 1996) i.e. one in which participants were randomly allocated to either a control or experimental group, and where the researcher manipulates an independent variable (in our case SERE participation) in one group whilst otherwise keeping the conditions the same. A consistent change across the experimental group (e.g. better grades for participants) would indicate that the independent variable (SERE participation) was responsible for the change.

In the first cycle (students taking their Masters degree in 2003-2004), we divided the volunteers into two groups. This allowed us to create both a randomly selected control group and a randomly selected experimental group, but only from within the participant sample. In post project feedback students who were in the control group...
felt disappointed they had not had greater access to SERE, while students in the experimental group expressed a great deal of satisfaction with it. As a result we did not divide the participants in the later cycles and all volunteers were given an equal opportunity to participate. This meant we were no longer conducting a controlled experiment, and instead had a quasi-experiment.

In the second and third cycle (particularly the third cycle) more use was made of all the students (participants and non-participants). All students provided data on their experience in both survey form and interviews and participants provided additional data on the usefulness of the various tools presented to them. While the form of this data collection was similar to the first year (though interviews were added) the lack of a control and experimental group made participant data more valuable. Data collected from both groups also allowed some research questions about users’ attitudes to software engineering for research to be examined in a quasi-experimental fashion.

Basili (1996) notes that an experiment can be performed on people with differing background and experience with relation to the experiment’s context. As there are no well-known approaches to the process of computer science research we do not regard this as a significant factor. Students’ experience with programming and specifically experience in industry were collected through a survey in the introductory lecture (in all three cycles), and comparisons to past experiences were solicited in the post project interviews in the final cycle.

Although the sample is self-selecting, by not restricting participation we got a large sample and a mixture of participants (with regard to abilities and experience) and projects (with regards to research methodology, supervisors and areas of computer science under investigation). This closely mirrors the real research environment in an academic department. The use of three cohorts of students, with about 50% choosing to participate each year, was at the upper end of feasibility for this research; however, a smaller number would not have produced results of equal value.

6.1.3 Selection of Approach
The experimental subjects’ use of SERE is a revolutionary change. Revolutionary change occurs when we introduce a new approach, e.g. “the proposal of a new method
or tool to perform software development in a new way” (Basili 1996)). The approach to evaluating a revolutionary approach is though the creation of statistical or qualitative methods to measure its effect, applying the method to case studies, measuring and analysing the results and then repeating the procedure.

The approach we use to develop and refine SERE is an evolutionary one. In the evolutionary experimental paradigm, existing solutions are observed, better solutions proposed, the new solutions tried and the results analysed. The process is repeated until no more improvement can be made, for example “the study of improvements to methods being used in the development of software” (Basili 1996). With the exception of the personal process model (page 162), all our artefacts are based on existing approaches and tools, which we have added to or adapted to a new purpose. The repeated nature of the experiment also lead to the later cycles being an evolution of the prior approaches.

We use multiple types of data collection which allows triangulation. The use of triangulation allows us to confirm findings and gain greater depth of understanding (Patton 1990). The concept originates from land surveying, where it was shown that, given the directions from a location to two landmarks, the location could be accurately pinpointed at the intersection. The confirmation benefit in such a scenario would be the use of additional directions to other landmarks to confirm the location. This could also be used to remove, or at least evaluate, measurement inaccuracies. Our multiple approaches to gathering data on the same topic act in a similar way give us a confirmation benefit. Our use of unstructured and open investigation techniques (such as open questions on surveys and in recorded interviews) provides the depth benefit to our investigation. It is possible for factors not taken account of through one approaches to be discovered with another method, for example the final surveys may miss something important which is then picked up in the post project interviews.

6.1.4 Selection of Intervention
All SERE interventions take the form of guidelines or recommended tools, both forms of process descriptors, which follow the RAISER approach. With the exception of the occasional reminder message via e-mail, the initiative is entirely with the students.
A website with access restricted to participants was used as the main delivery mechanism for process descriptors (i.e. tools, guides, and recommendations etc). Students chose which tools they wished to use. There was no requirement to use any of the descriptors and unlike other work that tests software engineering in the classroom (Humphrey 1998), the use of our tools was not tied to any of the students’ classes. Examples of tools included dOxygen (an open-source software package), a “getting started” document (a guide) and a research journal (a template). The address and login details for the website (available to students on and off campus) and the initial “getting started guide” were e-mailed to participants at the start of their project. This again lowered the initial effort required at the students’ end at a time when they were still unsure of their participation.

Participants had access to a software engineer (myself) via e-mail and meetings. The specific help available varied as the SERE framework developed, but in all cases queries related the experiment and the artefacts were answered by a single person. This reduced the chance of experimental noise that may have resulted, through varying interpretations of the artefacts, if multiple people been involved in answering questions. Intervention by others (not the subject or the experimenter) did occur and was largely the result of the supervisor carrying out their duties. This was not considered noise, but is rather seen as a key part of the environment. Supervisors, and in some cases other researchers, also occasionally provided observations: this was most formally done in the case of technical reviews (page 176).
6.2 Experimental Development
Our initial intention was to conduct a full experiment over the entire RAISER / RESET life cycle of a number of research projects. This was overambitious and rapidly scaled back at the end of the first year. After the initial cycle of results was collected our focus was restricted to the RAISER part of the process. The initial methodology called for a full experiment, the results from the first year caused us to rather choose a quasi-experiment design for the remaining two years. The difference between the full experiment and the quasi-experiment is a matter of random sampling compared with self-selection. We found self-selection to have distinct advantages in our experimental setup.

In the following sections we discuss the initial experiment design, followed by the design used for the final two years. In developing our approach to better assist researchers we created and refined a Meta Model of the Research Process, this model and its development is present as part of the introduction to each cycle.

6.2.1 The First Cycle (2003-2004)
At this stage we intended carrying out research on the full life cycle of a number of projects as they under-went RAISER, RESET and then a follow-on RAISER and RESET phase. After creating various plans for the experiment, a decision was taken to narrow the scope to the first RAISER phase only. The complete experiment was overly ambitious and not feasible within the limits of this research.

In this section we present the case study plan that formed the methodology for the first cycle. We explain the process descriptors used in this and subsequent years. Finally, we review the first years’ experiment.

An over-view of the plans made for the larger experiment (including RESET) are outlined at the end of this thesis in the further work section. Copies of the artefacts created in this cycle are presented in the appendix to this report.

6.2.1.1 Summary from the Meta Research Point of View
The first year saw us facilitating researchers by making existing tools and primary process descriptors. Figure 12 shows the relationship between the research, process
descriptors and software engineer. The diagram shows a set of research processes each of which contain one or more artefacts, and a software engineer who is creating these artefacts. The key fact differentiating this cycle from the later ones is that approach is not thought of as a process. As such, the process descriptors are considered tools to use, not descriptors of the research process or descriptors of its sub processes.

Figure 12: The first year process

6.2.1.2 The Initial Case Study (Planned Methodology)
This is the detailed case study plan for the first year. It was created prior to starting the experiment.

Aim: Explore the effect on the researcher of applying RAISER (or not applying it)
Type: Multi case, holistic
BSH type: Blocked subject-project study
Unit of Analysis: A researcher
Hypothesis:
- RAISER is seen as beneficial by researcher
- Effort spent in RAISER is seen as being for the benefit of the researcher
- Effort expended in engineering is minimal compared to other projects
- Researcher encounters fewer problems or can recover from them better
- Focus on publishing is improved
Selection of Pilot Projects:
Volunteers would be requested from the MSc class. Those who volunteer are divided into a control and experimental group. Both groups are provided with information on how to do software engineering relevant to research, and have access to recommended tools. The control group is left with access to the given information, whilst the experimental group has meetings where their progress and use of the tools is discussed. The experimental group is strongly encouraged to use the tools and methods provided, and recommendations to use tools are worded as a mandatory part of their participation. The division into groups is random, but the sampling is checked to ensure a mix of nationalities and academic abilities.

6.2.1.3 Implementation of SERE – the First Cycle
The first contact with students took place as a guest lecture in their software engineering course. The lecture introduced RAiser / RESET. It also examined the problem of over-engineering which wastes the limited time available to researchers, and conversely of under-engineering, in which research code is not engineered at all. This introduction aimed to raise awareness of the problems and to explain the RAISER approach of using minimal engineering, but doing it in a way that gives immediate benefits for the researcher. During class, students filled in a short form on their background, opinions on software engineering, and why they wished to participate / not participate. The form “MSc Introduction Survey 2003-2004” can be seen in the appendix (page 231).

As per the case study plan, we split the volunteers into an experimental and control group. Members of both groups were given individual passwords to the SERE website. Non-volunteers from the class were not given a password, but were told they would be asked to complete a couple of surveys towards the end of their MSc.

Initial meetings took place with participants (control and experimental) where they explained their project. A “getting started guide” (see appendix, pg 247) was given to the experimental group students and introduced them to the research process and the importance of keeping records e.g. in a research notebook (see appendix, page 250). The ideas in the getting started guide were elaborated in meetings with the students.
The document was also put on the web so control group members could look at it if they wished. In the “getting started” guide students were asked to pick potential conferences and journals in which their work could appear.

The limited use of industry style software engineering in a research project (involving at most a couple of researchers) was discussed with students in early meetings. Students were told that the exception is a high level architectural design. E-mails were sent (to all participants) encouraging them to create such architecture diagrams.

A second set of meetings took place as part of a student’s regular supervisory meeting. This was largely an observation, though suggestions were given on software engineering topics and where possible on research approaches and related work that could be useful. This simulated the input a RESET laboratory might have into new projects.

Coding standards (see page 251) were released on the web and notification mailed to students as well as it being mentioned in meetings. Those in the experimental group were asked to read through and start applying the coding guidelines and then book a technical review to examine their implementation of it.

The coding standard stressed the importance of commenting and a recommended style was provided. The guidelines aim for more readability and asked that comments conformed to a standard suitable for use with the dOxygen tool.

Students were encouraged to document (in comments) any approaches they had tried and rejected in their work. Examples given to them included the use of data structures and algorithms that were tried and discarded after significant investment, such comments act as a reminder for later report-writing and may prevent others wasting time in the future. Where something was implemented without a critical decision being made, this information was also to be considered useful. Students were told that a lack of such rationale behind a decision might indicate an opportunity for future research or the opportunity to recode for greater efficiency.
While still half finished the code was supposed to receive a Formal Technical Review (FTR). FTRs were meant to highlight weaknesses that may lead to a loss of knowledge, wasted effort, or other inefficiencies by the current or future researchers. The idea was also to detect and comment on defects, and to improve the process of development. More details of the Technical Review plan can be seen in the appendix (appendix, Section 12.8.1, page 271). Unfortunately only one student in the first year had a review. Other students scheduled reviews then put them off and finally ran out of time.

All students were given a survey after their project, the survey “MSc Final Survey All Students 2003-2004” can be seen in the appendix (appendix, pg 234). Participants were given an additional survey (appendix, page 236) with some questions being directed purely at the experimental group. Supervisors were also given a very short survey, which was related to the performance of the student and the value of reusing their work based on the nature of the work and the quality of the workmanship.

Non-volunteer surveys were aimed at detecting leakage from the volunteers in terms of methods and tools. A number of questions were asked of both volunteers and non volunteers to further develop understanding of the research environment, the problems students faced and research experience. This was intended to provide a comparison between those who were involved and those who were not.

Softcopies of the students’ reports and code were collected, along with the students’ results for the project and for the taught part of the course. This was later analysed.

The Research Case Study was, we believe, implement as described in The Initial Case Study as described on page 96.

6.2.1.4 Influential Factors

Change of scope
According to the plan at this point in time, a number of projects were supposed to be RESET. The projects were to be chosen based on both student and supervisor feedback on the likelihood of the project being reused. The decision to focusing only on the initial RAISER part of the research means some of this data was not collected
in later cycles and the first year results have a slightly different context to the latter years.

**Environment - Resources**
Without a policy to support process improvement, discussions of what makes a good process are entirely moot (Ward, Fayad et al. 2001). The organisation, in this case a department, must have in place a policy to support process improvement. This includes allocating sufficient staff and resources to assist with this task.

In the case of this research, time was given to introduce the concepts to MSc students. Web space and other resources were available. The project emphasised the use of free software, which removed the financial cost of purchasing software. The MSc mailing list was available. Most importantly, the department allowed MSc students to participate in the experiment. In most cases the support was from supervisors and was passive. However, if this experiment had been disruptive, students would have of their own accord (or due to encouragement) dropped out or refused to participate. That this did not happen was encouraging.

### 6.2.1.5 Reflection after the First Cycle

**Scale**
The first cycle provided an exploration of our research area. The most obvious finding is that this is a very large research area with little prior work. The experiments we planned by their nature take a lot of resources and occur on a large scale. In order to be manageable, the scope of the experiment originally planned needed to be drastically narrowed. Originally other options were considered, e.g. while ideally each research project should be RESET by a team, resetting by a single person (myself) was considered. Likewise the ideal plan for resetting 8 projects at the end of phase one was reconsidered and plans made for reducing this number. Even with a reduced number of projects scheduled to be RESET, and without the cost of a team, such reset work was not feasible within the time constraints of this project.

The value of the data collected was also questionable as to fully investigate the lifecycle would require a number of projects that are started, finished, reengineered and passed on to a new researcher. The new researcher’s interaction with the RESET
product would then be evaluated, as would their final output. To get accurate results would require that this happens for multiple projects. Unless conducted on a very large scale and over an extended time period, such an approach is likely to fail due to a lack of projects that meet the requirements.

In re-scoping the investigation only the initial RAISER phase of the research, that is researchers creating a new product, was to be considered. As part of this we considered issues related to reuse, both in terms of the students reusing existing code, and in terms of the code produced being reusable. A RESET process was, however, not carried out on any of the code.

The concept of a software development lab was tested out in other ways. Code reviews though not successful in the first year, we still included within scope. In planning them it was noted that reviews required the reviewer (myself) to become familiar with many projects within a short period of time. Many of these would be in unknown research fields. This is largely similar to the task a RESET Engineer would face in supporting academic research, or as the first step to Resetting a product. The “read everything in two hours” design pattern was developed to help with this task and may provide useful for RESET work in general.

In this work we are not merely doing an experiment then testing the result, but rather continuously monitoring the progress of the experiment and adjusting it to get to the right result. This result is about more efficient research, and the goal for our research after the first years was to develop and refine the approach to get there. This is an acknowledgement of the evolutionary nature of the research. The new scope would allow a greater focus on refinement in the later years.

Involvement and need for training
Volunteers were interested in the work but naturally their research comes first. This is a problem for the research, but is an accurate simulation of the real conflicting priorities of all researchers. One solution to this problem is training prior to starting work on the actual project. This is a solution both to the experimental problem and the problem in the real world. For researchers to improve their approach there is a
requirement to research consciously rather than subconsciously. This requires a new way of thinking and training that is more than learning by osmosis.

**Communication**

Interaction with MSc students and their supervisors needs to be more organised with checklists for each case (project / student) listing the things that have to happen as part of the case and preferably listing rough dates.

Too much responsibility for participation was given to MSc students (in the experimental group). While this did accurately model the way most research projects are run, the compressed time scale means some things did not occur e.g. no one sent in a list of journals / conferences. Other things occurred late, e.g. meetings with supervisors and students (in fact in a few cases these did not occur at all), FTRs (all occurred, but in some cases were postponed and in all cases occurred too late in the process), and surveys (even with survey one, two thirds were submitted late; the final survey took a lot of follow-up work). More regular contact would heighten the priority given to the experiment. This would have the side effect of making participation more burdensome, but given participants’ feedback, this could be increased and still be within acceptable limits.

One thing that did work well was requesting students’ non-university e-mail addresses at the start. This meant that people could be chased for the final survey, despite having left the university and stopped using their university e-mail addresses. Where personal reminders were sent out the response rate was very high.

### 6.2.2 The Second Cycle (2004-2005)

From the second year the focus was restricted to the RAISER process. Based on feedback from the first year, further process descriptors were created to overcome some of the difficulties and barriers to adoption.

In this section we present the case study plan that formed the methodology for the second cycle. We explain the process descriptors used in this and subsequent years. Finally we review the second year’s experiment.
6.2.2.1 **Summary from the Meta Research Point of View**

The second year saw the key change of meetings and assistance also being classified as a process descriptor. Assistance and access to all process descriptors was available to all who signed up to participate. Secondary process descriptors, i.e. those whose sole function is to aid in the selection or use of other process descriptors, were also created. The overall process became much more lightweight and did not involve supervisors.

Where the first year made tools available, the second sought to lower the burden of adoption and maximize the benefits from tool use. Rather than a collection of tools, there was now a process view for the software engineer. This enabled intervention in a timely fashion. While researchers knew there was a process (the process descriptor template) they were not managing it and only tailored it in so much as they opted in or out of various sub-processes along the way. This is shown in Figure 13. The use of the UML inheritance notation signified that researchers were not simply including process descriptors, but now adapting them to their own use. This was encouraged from day one of the second cycle.

![Figure 13: The second year process](image)
6.2.2.2 Second year Case Study (Planned Methodology)

The case study plan is the same as the first year, but with the goal of a focus on publications removed. From the first year’s experience it does not seem relevant to these researchers. The major change is in the selection method of participants, and the increased focus on the benefits to the current researcher.

**Hypothesis removed:**

- Focus on publishing is improved

**Selection of Projects:**

Participation is self-selecting. A case of the benefits is presented as well as an indication of the cost of participation (completion of surveys and a couple of interviews, everything else is now presented as optional).

6.2.2.3 Methods Implemented

Those who volunteered would all be in the experimental group. Non-volunteers from the class would not have access to the site, but would still be asked to complete surveys as before. The in-class survey was changed to gather students’ initial opinions on software engineering and research before the presentation discussed these topics. The students were given a survey (see “MSc Introduction Survey 2004-2006”, pg 232) with the questions blanked out. The questions were presented in the slides to prevent bias in responses from the text of later questions. Towards the end of the presentation, feedback from students in the first cycle was shown to the second cycle students.

In this cycle, all the resources from the first cycle were available from the start. New tools were largely secondary process descriptors, i.e. documents, guidelines, examples to help students evaluate the usefulness of tools for them, as well as the costs and benefits from using various tools.

New secondary tools were created, including; example input and output files for dOxygen, (an edit version of an earlier student’s work, used with their permission), and a “Technical Review Preparation Guide” (in appendix, pg 270) that supported technical reviews and mentioned dOxygen, journals, and other tools.
The recommendation for students to selected journals and conferences their work could be published in remained in the getting started guide, but was not mentioned to the students (in contrast students were asked to provide their list in the first year and none did).

Initial meetings took place as before, as well as final meetings. These were recorded with the consent of both parties. Due to the shift in focus, meetings with supervisors were not included.

Softcopies of the students’ reports and code were collected, along with the students’ results for the project and for the taught part of the course. This was later analysed. Where available dOxygen output was also collected from student websites or CDs included with their thesis.

The Research Case Study was implemented as planned in the Second year Case Study (pg 104).

**6.2.2.4 Influential Factors**

With the exception of a large-scale change to the research and experimental space, the environmental variables remained fairly constant.

The department moved offices at the start of this cycle. This had a large environmental impact. MSc students were no longer located close to research staff. I was also no longer in the office I had previously been operating from (located next to the MSc lab), and in fact was located some distance away in the basement of the new building in a shared open plan office with over 15 other researchers. The office was behind 3 automatic locking doors. This decreased visibility and communication with students, greatly reduced the chance occurrences that initiated much of the discussion in the first year’s group, and made meeting students far more awkward as a suitable space had to found – meaning a follow-up meeting was usually in a different location. In some cases the old office was available, but usually not. A general impression from staff was that this cycle’s students were less communicative and less engaged. This could well be an effect of the changed environment and the separation between MSc students and the rest of the department.
6.2.2.5 Reflection After the Second Cycle

The first year provided the framework from which the experimental techniques could be evaluated and refined. The second cycle not only refined the scope but also significantly shifted the focus. Limiting the research to the first part of the RAISER/RESET process allowed greater focus on the needs of researchers. The process framework emerged over this cycle, which in turn added a new level of insight and placed the previous year’s work in a stronger context.

The methodology for conducting the experiments was further developed, but also stabilised and refined. This is consistent with the process approach of continual improvement. In an early report on this work we stated that “this steep learning curve will continue until a selection of the current projects have been re-engineered and the approach for monitoring this re-engineering has been fully developed”. Since then the re-engineering was placed outside the scope of this research, but we identified that the learning curve on the research side is on going. The process of creating useful process descriptors (tools, guidelines or recommendations for part of the research process) can be applied to an infinite number of tasks. In this research we are limited to the more generic tasks and to those that seem to offer the greatest benefit… the limiting factor, however, is time, not the number of tasks to be looked at.

We previously noted the “recursive nature” of this research. The literature has referred to this as a “doubly indirect nature” (Osterweil 1987), a more accurate description. Osterweil suggests that “the processes which we intuitively carry out are more complex than might be expected and [this] explains why it is often so difficult to explain them to others, so easy to overlook unexpected consequences, and so hard to estimate the effort needed to carry them out”. These problems we had unknowingly been tackling became evident this year as helper descriptors were introduced, making adoption easier and more widespread amongst participants. Adding clarity and helping researchers understand the costs and benefits of a process change is a critical factor in enabling adoption. Tools are relatively easy to find, but other processes (such as how to start doing research) are more individual and harder to capture. Our data has shown this is an area where many of our research subjects (being new researchers) want help. It also explains why the “Getting started guide” was (according to the cycle one MSc students) the most useful tool in the first year of the experiment.
In the reflections on the first cycle we noted “a requirement to research consciously rather than subconsciously”, from this years experiment we add the need to formalise this (at least at a high level, similar to what we suggest student use in their architecture designs). Students were introduced to dOxygen in their initial meeting and sample input and output was made available to allow researchers to better assess the value of this tool. This is however not enough. For the final cycle it was decided to take this a step further to assist students in creating their research process in the form of an OO diagram / set of dOxygen comments. This is the next step in assisting researchers to research consciously and in a more systematic manner.

We noted in the reflection on our first year how “we are not merely doing an experiment then testing the result, but rather continuously monitoring the progress of the experiment and adjusting it to get to the right result”. This statement is still true after the second year. While most adjustments have been in the nature of additions or modifications, in one case we now recommend against a tool previously put forward. The tool in question is CVS which for those new to it is too much of a burden.

The result we were aiming for after the second year remained that of more efficient research and the goal of our research was “to develop and refine the approach to assist researchers get there”. The emphasis signifies a change based on the experience of the second cycle and growing understanding of the nature of the research environment. The change reflects the realisation that each process must be unique and the researchers themselves must be responsible for it. The most Software Engineers can do is develop ways of assisting researchers. The research process, like the learning process, belongs to the subject, be they a student or a researcher.

6.2.3 The Third Cycle (2005-2006)
In the third year a more researcher-driven process approach was created. This was a dramatic and revolutionary change. Only a few students participated enough to take advantage of this new approach. For the rest it was as if they had taken part in a previous cycle of the experiment.
In this section we present the methodology for the third cycle and review the third year’s experiment.

6.2.3.1 Meta Research Point of View

In the third year an attempt was made to shift the researchers’ focus from process descriptors to a custom process that includes various descriptors. In the first year there were tools, in the second helper tools lowered adoption and made the available tools more useful (for less effort), in the third students took charge of developing their own research process. Some students in the third cycle did not get beyond tools, others stopped at the helper tools, and some adopted the new way of thinking and began integrating their tools and approach. A software engineer cannot improve a researcher’s process, just like a teacher cannot make a student learn. The best an engineer can do, is to make knowledge available and facilitate the researcher in their own process of process improvement. Such improvement requires that the researcher be critically aware of their process and choices and systematically approach their task. This perception is shown in Figure 14. Although only included in a very limited way (through Technical reviews) an element of group support was also added. This need further investigation and development. The final process can be seen in Figure 15.

![Figure 14: Process Improvement Process](image-url)
In the final process the intention is that the researchers step out of the process and take control through critical reflection and planning. The process descriptors now include an overall research process plan for each researcher. In addition discussion between researchers is meant to help them form ideas and reflect. Note that each of the peers would be in the position of the researcher being critically reflective as well, but this is not shown for simplicity.

6.2.3.2 The third year Case Study (Planned Methodology)
The case study plan is a development of the second year plan. The goal of focusing researchers on a process approach is added. Also added is the goal of exploring the use of a personal process model (pg 162). The new way of thinking is a significant change in perspective.

Additional Hypotheses:
- Researchers adapt to process thinking and “take charge” of their process

Additional Aim:
- Explore the use of personal process modelling in a research environment context

Selection of Projects:
Participation is self-selecting, and like the second cycle students choose how much they participate and which aspects they adopt. There is significant coupling between
process plans, documentation guidelines and technical reviews, this may have an impact on participants electing to either use them all, or none of them. Again this is the students’ choice.

6.2.3.3 Methods Implemented
The third year progressed much like the second year, except that some of the participant researchers now took a process perspective. This led to wider adoption of technical reviews, and multiple technical reviews taking place on each project. Process plans were also introduced to the students. These were created by students from a process descriptor (that took the form of a template) and themselves subject to review.

The process plans were created using dOxygen comments based on a template provided (pg 273). These were then generated into dOxygen output (pg 281) and shared with reviewers prior to the review. The three aspects of review included the code, the process plan, and the documentation. These were combined into one, two or three reviews depending on the progress and needs of the individual researcher.

The third year saw a much greater use of interviews as a means of data collection. This was an attempt to correct for the low involvement of students in the second year and also a failure of students to return surveys. Interviews were scheduled after the student’s demos and the scheduling often took place not with the student but with their supervisor.

6.2.3.4 Influential Factors
The environment for the third cycle remained largely as it was for the second year. One notable change was that posting rights from my mail account to the MSc list seemed to have been removed, all e-mails were approved by the system administrator but this may have led to delays in delivery and the extra information appended to the subject may have made these e-mails (e.g. surveys) seem lower priority than they were in the past. Some building works also occurred outside the MSc lab late in the project. The added noise may have been a distraction for students at times, but is not seen to be a large factor as part of the MSc space was fairly isolated from the noise and students relocated themselves to this space.
6.2.3.5 Reflection After the Third Cycle

The third and final cycle saw the introduction of the personal process model. This was used by three of the students, a small number of case studies for measuring the impact of the tool, but a significant number in terms of their feedback.

The personal process model was the most revolutionary of our approaches and a significant change in perspective was observed in the students who used it compared to the rest of the participants and non-participants. Positive participant feedback and observations of improved communication such further work in this direction would be valuable.

The documentation guidelines were followed more carefully by more students than had occurred in the past and dOxygen was made more use of and almost became a standard inclusion in students’ electronic submissions.

Reviews were also more widely adopted, specifically by those adapting to a process perspective. In one case a supervisor assisted in a review. In another a friend of the researcher (themselves a PhD student in the area) observed the review and provided confidential feedback on the review and their perception of researchers’ attitudes to it afterwards the review. In one case the process plans led to a researcher recording publicly details that perhaps should not be public (e.g. reflections on supervisors). These were however caught in review and removed from the process plan.

The use of interviews provided a very rich data set. Due to the volume of data only partial transcriptions were made (extracting interesting and relevant information). The interviews added a greater depth of understanding about the students, their experience and their approach to research.

The third cycle demonstrated a maturation of the approach and a greater depth of adoption amongst some participants. The various components now formed a cohesive whole not only to external observation, but also to many of the participants.

While a fourth cycle was not carried out, a number of the students (mostly participants) from the first three cycles are now PhD candidates in the department. At
a departmental talk where the personal process model (including developments as a result of feedback from the third cycle) was presented, former participants were both interested enough to attend, and showed continued enthusiasm and support of the approach. One participant commented that a similar talk introducing the personal process model to all MSc students would be of benefit before they start their projects. Similar comments about the need to make the tools available to post third cycle students were made, and discussions started with the department about making the research assets more generally available. This demonstrates a high level of acceptance and a very positive outcome for the final year of experimentation.

6.3 Conclusions
In this chapter we have presented both our overall design choices and those choices made as the experiment evolved. Various factors discussed in this section should be kept in mind when interpreting the results in later chapters, and particularly when comparing between cycles.

The move to a purely self-selecting sample was both necessary and positive in developing the experiment. The increase in data collection (including from non-participants) provides a richer idea of the research context despite the self-selection. Other factors such as the improving maturity of SERE process descriptors, combined with the change in experimental design, made the first year results difficult to compare to the later years. This is a result of substantial improvements in the later years and is regarded as a positive development for the experiment.
7 Experimental Results

Surveys were used at the start and end of each cycle to gather feedback on the participants, their research experience and their views on software engineering in general and on SERE in particular. Some surveys were paper-based while others were collected via e-mail. A case study approach was also used, and meetings and technical reviews were recorded and analysed along with observations. This chapter presents a summary of these results.

7.1 Initial Surveys

Surveys were used to collect data on the MSc students before they were introduced to the project. The method used varied between the first and later cycles, but in both cases the surveys aimed to collect data on students’ backgrounds and preconceptions. Some of this background information has been included in case study data in the next chapter. In the second and third cycle the initial survey was used to collect information on students’ initial impressions of software engineering and their expectations of the impact software engineering could have on research. Students’ motivations for signing up or not signing up were collected in the initial survey in all three cycles.

In this section we discuss the sample of respondents, followed by a discussion of the background to those who replied, a discussion of their pre-conceptions about software engineering and finally we examine their initial thoughts on participation.

7.1.1 The Samples

In the first cycle students initially expressed interest by filling in a text-based form which was e-mailed to them. After a few reminders, 16 students completed the form out of a class of 30. A further two students replied to the e-mail but did not complete the form; one of these expressed concern about the amount of time participation would take. This supports the observations made in chapter 2 regarding the need for minimum overheads in SERE.

In the first cycle an additional signup step was needed, this involved paper forms that were distributed via the course secretary. The number and timing of reminder e-mails was moderated on the conservative side so as to not create hostility to the experiment
on the part of the students. 15 forms were received; three of these were from students not wishing to take part in the experiment and 12 from those wishing to participate. Results examined later in this thesis show that some students (in other cycles) chose not to participate because they were unwilling to consider anything that would take extra time – regardless of potential benefits. This may in part account both for their non-participation and non-return of the survey.

In the second cycle paper forms were distributed during a lecture given to the class in November (well before they selected their projects). The forms (appendix, pg 232) were collected at the end of the session. 27 forms were received; four of these were mostly blank. The forms asked for an expression of interest, the final sign up was done over e-mail.

In the third cycle, paper forms were again distributed in a lecture, before projects were selected, and were again only an expression of interest. The same form was used as in the previous cycle. 18 forms were received; one of these was mostly blank with an apology for being late.

In the next section we examine the make up of each sample.

7.1.2 Background of the Students
In an attempt to find out about students’ previous level of experience with software engineering, an open response question asked them to describe their “software engineer experience / training”. This question proved to be little used. In the first cycle, all respondents bar one listed their degree. The student who did not list their degree wrote that they hadn’t worked yet and hence had no experience. This was an international student and the differing response is thought to be a result of differing cultural understanding of “experience / training”. The only interesting details collected from this question was that four of the students had degrees with software engineering specialisations.

A second question, “have you worked in Computing, IT or related jobs”, proved to be more useful. Of the 11 participants in SERE in the first cycle, four had prior work experience. The experience included: System Administrator (two years), R&D
software engineering in the banking industry (one and a half years), Systems Engineer in the Nuclear Power industry (one year), Systems Engineering developing a driver for a hardware card (one year). Of the three not participating, one had experience as a university Teaching Assistant. The other two did not have any relevant work experience.

In the second cycle ten students had industry experience and nine did not. The remaining students did not answer this question. In the third cycle, seven had industry experience, nine did not, and three did not answer the question.

The information on students previous work experience is provided as part of the case study data for students this research discusses in more depth later in the thesis.

7.1.3 Preconceptions on Software Engineering

In the first cycle, students’ preconceptions on software engineering were not collected. In the later years the initial survey collected this information. Students were asked to define software engineer, whether software engineering could help research, and what the results of a lack of software engineering might be.

When asked to define software engineer the most common response included the idea of “the process of developing software systems”. In the second cycle there were 15 responses that included the word “process” in this context. In the third cycle there were 10 responses along these lines. Two students did not answer this question in the second cycle, and four in the third cycle. Other responses may be found in the appendix (pg 291) and highlighted issues already discussed in Chapter 4. The predominance of the process paradigm within students’ answers (gathered prior to their introduction to SERE) provides validation of the appropriateness of the process approach in SERE.

7.1.4 Benefits of Software Engineering

Building on students’ understanding of software engineering, students in the second and third cycle were asked if software engineering could help research. Students were then asked what problems might arise if software engineering was not used. These questions separated a lack of faith in software engineering in general from environmental elements related to the student and their project. Another potential
factor is students’ impression of SERE itself, this is accounted for by gathering these initial impressions of software engineering for research, and students willingness to try a new approach, before SERE is introduced. A detailed explanation of SERE was not provided in the class and instead took place one-on-one in an initial meeting with each volunteer.

When asked if software engineering was of benefit to research, only four students in the second cycle said no, and 21 said yes. In the third cycle, only 5 students said no, and 11 said yes. This showed that students (regardless of their views on SERE participation) largely thought some form of software engineering would be of benefit to research. This is a strong contrast to the lack of research into software engineering for the academic environment, and the limited use of software engineering by academic researchers.

**Software engineering is not helpful**
The students who said software engineering was not appropriate for research expressed reservations similar to those expressed by some academics. These concerns included:

- Unknown and changing requirements
- Concerns that a set process may restrict exploration and creativity
- A view that once the research is over, then software engineering would be useful, during research the overhead is too high
- A concern that software engineering could create “better developed software” but “not necessarily… correct software”.

The repetition of these themes in both sets of students, as well as from academics themselves highlight problems that could arise from approaches that are not suited to the academic research environment.

**Software engineering is helpful**
Students who said software engineering could help research also justified their position. Common ideas from students in the second cycle on why software engineering would help research included: better understanding, reduced costs, better time management, help solve problems faster, “includes tactics that can apply in any
kind of software”, helps make research software efficient, helps establish requirements, helps avoid mistakes, and that it “provides a systematic method of dealing with problems”. In the third cycle, those who said software engineering could help research expressed similar views to those in the second cycle. An additional idea of software engineering as failure prevention also emerged.

More detailed explanations are provided in the appendix (pg 291), a core theme, however, is the provision of a systematic approach – though students differed in the degree of formality they expected this structure to take. The benefits students expect generally compare well with those previously discussed in chapter 4 and summarised in that chapter’s conclusion (pg 75).

The idea of specifications and requirements, and the idea of a predetermined process were given by some students as examples of benefits, and by others as causes for concern. The SERE approach promotes flexibility and agility over rigid specification, in effect agreeing with those who reject software engineering, but claiming our form of software engineering, being tailored to the research environment, does not suffer from these problems. While some see software engineering as carrying risks, not using software engineering also has risks.

### 7.1.5 Dangers without Software Engineering

In both cycles, students worked in pairs to compile a list of problems that could occur if software engineering is not used in research. This was followed by a class discussion where pairs discussed their answers with the class. As students added additional answers from the discussion, repetition of answers to this question has no meaning. Each unique answer, however, contributes a new idea. The answers have been divided into eight broad themes and are presented in Table 5.
<table>
<thead>
<tr>
<th>Category</th>
<th>Dangers given in cycle two</th>
<th>Dangers given in cycle three</th>
</tr>
</thead>
</table>
| Non-systematic process         | • No structure in research techniques  
• Knowing when your research software is complete is difficult                              | • Process might be chaotic  
• Chaos in research might follow due to lack of organisation  
• Harder to follow progress  
• Difficulties predicting future workload  
• Might go down a wrong path  
• New requirements may arise at the end |
| Time delays                    | • May take longer to develop  
• Can't finish on time  
• Delays  
• Waste much time on system faults                                                  | • Takes longer  
• Hard to meet due date  
• Poor time management                                                                |
| Reuse & adaptability           | • Unorganised code, so it will be difficult to update the program  
• Reuse  
• Lack of flexibility - changing the software can be hard  
• Difficult to analyse and do future work  
• Similar projects in the future will have to be developed from scratch              | • May have poor architecture / little structure  
• Not possible to extend functionality  
• Complications later on  
• Difficult for other participants to enter the project                              |
| Documentation and communication| • Lack of documentation - another researcher cannot continue the research easily  
• Demonstrating your methodology to others may be difficult  
• Inability to communicate the work                                                | • No documentation  
• May not be able to communicate the design to another team  
• Inadequate documentation  
• Lack of documentation makes writing report difficult  
• May not be able to communicate the design to another team                          |
| Efficiency                     | • Inefficiency  
• Lose your way                                                                    | • Inefficient use of resources  
• Higher cost                                                                         |
| Product quality                | • Reliability  
• Compatibility                                                                  | • Not able to scale                                                                      |
One student noted that, “Many of the reasons for using SE are not relevant to research projects”. Despite the large list of items their colleagues came up with (shown in Table 5) this student does have a point, items such as a lack of a detailed specification are in and of themselves not a problem in the same way they might be in industry. New issues such as replication of results, however, mean existing approaches may be used to achieve new benefits. While a few of the items in the table are simply a part of the research environment, other items can be seen as problems which SERE should mitigate against.

Having created a list at the very start of year two and year three it is, perhaps, surprising more students didn’t participate. Part of the reason for this is that students still felt their projects were still small enough to be manageable without a software engineering approach. Others simply did not want to get involved in additional work. Students’ thoughts on participation at this initial stage are discussed in the next section.

### 7.1.6 Thoughts on Participation

Students were asked whether they were interested in participating in SERE, and what their reason for participating or not participating were.

In the first cycle twelve students wished to participate and three did not. Reasons given for participating by students in the first cycle included:

- To help with the dissertation
- To properly engineer the project
• To learn new skills
• To help with the experiment

One student said they “felt pressured [to participate] by department e-mails – there were loads of them” and that they “felt guilty not to”. Those students who chose not participate were mostly concerned about the time it would take. A full break down of answers is available in the appendix (pg 290). Only two students were influenced by their peers, and both these students expressed interest in participating.

On the second cycle 16 students indicated that they would like to participate and eleven indicated that they would not (four explicitly and seven by not completing the participation section). Those students who wished to participate largely wanted the experience and the opportunity to learn. This motivation was repeated in the third cycle.

In the third cycle ten students indicated that they would like to participate and four indicated that they would not (two explicitly and two by not completing the participation section). A further two students indicated that they could not participate as they were part-time and would not undertake their project until 2007, commented that they really would like to participate and were disappointed this was the last year, both of these students filled in and signed the participation section. One student was undecided. Student who wished to participate largely wanted the experience and the opportunity to learn. The undecided student was concerned about the obligation, what help would be provided and whether they could leave part way through. One of the students who said no commented, "I'd rather not put the time investment in and have to spend time learning new techniques".

### 7.2 Final Survey - All Students

A survey was sent via e-mail to all students as their projects were finishing. The survey reflected on students research experience, their research process, and the reusability of their finished code. The results are present by cycle to aid interpretation.

They surveys collected data on various factors which are presented in Table 8 and Table 9 for the second and third year. Similar tables for the first year are available in
the appendix (Table 28 on pg 294, Table 29 on pg 295, Table 30 on pg 295). The columns in all these tables can be explained as:

<table>
<thead>
<tr>
<th>Others code:</th>
<th>Was existing code used in creating the project?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERE participant:</td>
<td>Was the student a participant in the experiment?</td>
</tr>
<tr>
<td>Extended:</td>
<td>Does the student feel their code can be extended?</td>
</tr>
<tr>
<td>Extended Comment:</td>
<td>Additional thoughts on someone extending the code</td>
</tr>
<tr>
<td>Easy to use:</td>
<td>Does the student feel their final code is easy to use?</td>
</tr>
<tr>
<td>Journal:</td>
<td>Did the student use a journal?</td>
</tr>
<tr>
<td>High level diagrams:</td>
<td>Did the student use a architecture or other high level diagram</td>
</tr>
<tr>
<td>Source Control:</td>
<td>Did the student use source control, e.g. CVS</td>
</tr>
</tbody>
</table>

7.2.1 The First Cycle

There were 8 experimental subjects. Of these five (the majority) used other people’s code, and three didn’t. Half the students felt their work could be extended. Within the experimental group there was a variety of practice (as shown in the appendix, Table 28 on pg 294) with some students appearing not to use the tools provided.

There were eight control group subjects. Half used other people’s code. Five felt their code could be extended and three that it couldn’t. Again a mixture of responses, but only seven positive responses in total to the questions on journal, high level diagram, and source control use (which represent SERE recommendations). This compares with 11 in the experimental group, which had the same sample size. This suggests active participation in the experimental group does increase adoption compared to simply having access to the tools website.

Five students chose not to participate in the experiment, but still completed an end of project survey. Two used other people’s code and three didn’t. Two feel their code could be extended and three that it couldn’t. The combined number of positive responses on journal, high level diagram, and source control use is only two. Even taking account of the small sample size (five, compared to eight in the previous two tables) this is much lower. This suggests participation increases adoption, compared to non-participation, regardless of whether students were in the active or control group.
The result showing that participants were more likely to use a journal, high level diagrams and source control are perhaps not very surprising given these things were explicitly mentioned to participants and perhaps not to the non-participants. If students were not already computer science graduates, and had they not also each had a supervisor this result would not have been surprising at all. By contrast, the results suggest that that these basic research and engineering ideas were possibly not recommended by supervisors, or if the recommendations were made, students did not see this advice as important. Similarly, the results suggest that these ideas did not occur to students of their own accord, or again, if they did, students rejected them.

An alternative view on this data is given in Table 6 and Table 7. In these tables the data is divided based on whether student felt their work could be extended or not. In both tables answers showing that students have not used other people’s code, and that they have not used a high level design, have been highlighted.

In Table 6 those students who felt their work could be extended are listed showing whether they reused code, their group status (experimental, control, neither), their view on how easy it would be for someone else to work with their code, and whether they used a high level design.

<table>
<thead>
<tr>
<th>Others code</th>
<th>Participant Group</th>
<th>Easy to use your code</th>
<th>high level design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Experimental</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Neither</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Neither</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6 Projects students thought could be extended 2003-2004
In Table 7 those students who felt their projects couldn’t be extended are listed.

<table>
<thead>
<tr>
<th>Others code</th>
<th>SERE participant</th>
<th>Easy to use your code</th>
<th>high level design</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Experimental</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Experimental</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 7 Projects students thought could not be extended 2003-2004

Using this view there appears to be a correlation between projects where students felt their work could be extended and both their reuse of code and their use of a high level diagram. Likewise a lack of reuse and high level diagrams seems related to students saying their work cannot be extended.

Another view on the data is based on the use of a journal. All of the six people who kept a journal reused other people’s code. In all but one case they feel their code can be extended. There may be a missing factor relates to meta-effort or a more systematic approach, i.e. effort spent on the project but not on coding. This is a more likely explanation that the journal itself having this impact. The people who did not use a journal provided a mixed set of results in terms of their reuse of others code (five yes, ten no) and in terms of their thoughts on extending their code (six yes and nine no). As shown in the previous tables, the reuse of others code and the view on extending ones code seem related to each other rather than to the use of the journal.

### 7.2.2 The Second Cycle

Only four students returned the final survey in the second cycle. All four were participants and also returned their participant survey form. One major difference between the first and second cycle was the recommendation not to use CVS. This was a result of feedback from the first cycle and various related to configuration. A file synchronisation program was recommended instead, though students were also
encouraged to use more manual methods such as creating copies of files or archiving them in a compressed format.

Table 8 shows the results from the participants who returned their surveys. The results show a possible increase in journal and high level diagram adoption, from 50% of experimental group participants in the first cycle to 75% of participants in the second cycle, though the sample is too small to place any certainty on this. What can be said is that from the participants at least we do see adoption.

<table>
<thead>
<tr>
<th>Others code</th>
<th>SERE participant</th>
<th>Extended</th>
<th>Extended Comment</th>
<th>Easy to use your code</th>
<th>Journal</th>
<th>high level design</th>
<th>Source control</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Not finished yet but when I do finish my project could definitely be extended</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Sync</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>I think so</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Certainly can be extended</td>
<td>? ²</td>
<td>Yes³</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>YES, it's going to be used as basic framework to build an industrial application</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 8 Final MSc Survey responses from all students 2004-2005

In open questions students were also asked what single thing would make it easier for others to reuse their code. The question was aimed at extracting weaknesses students observed in their final product and highlighting issues that would be detrimental to reuse.

Three students responded, and they said:

- “cannot answer at moment - possibly code documenting”
- “Create documentation that would explain all the different modules”
- “be more systematic and formal when I comment the code” thought this student also stated that their code was “well documented (internal comments, UML diagrams)”.

² The exact answer is “Easy once he reads the paper describing it and reading through the code, not easy as comments are insufficient”
³ The exact answer is “Yes, but with very little success”
These answers fit in the “non systematic process” and “documentation and communication” categories we previously identified in Table 5 (pg 119). Another problem in the category of “reuse” was evident in one of the students explanations of difficulties they encountered, the student explained, “I was planning to use the code of a research project [project name removed] for the rapid creation of my database. By the time I got the code I had 3 weeks left, there were no comments to the code and no documentation, so I had to build another system into the system to insert data”.

Asked if they would use more or less software engineering if doing the project again, the students all replied they would use more (one suggesting more or the same amount). One student commented that they would use “more - high level at beginning, however that is easy to say now as I have an idea of how things are most likely going to develop. At beginning I had very little idea”. This is again an inherent problem of research coding, but the idea of more high level design at the beginning can at least help clarify some of the ideas before coding is started.

One of the students included an extended comment in their response, prefixing it by saying, “there is an argument here I’m sure you will be interested in”, the comment stated: “Business code versus research code. Should we use software engineering in research code as we do in business apps. Obviously not because they are more throw away, do not need to be as robust, are not tested in the same way as business code. So should we use the same techniques of analysis and design such as requirements. In my project the requirement is … not very extensive and so should warrant less formal procedures - i.e. get on and code it :-)” This highlights the gap between rejecting overly formal and prescriptive approaches on the one hand, and taking an completely ad-hoc approach on the other. The aim of SERE is to fill this gap, taking into account the specific needs of research.

7.2.3 The Third Cycle
Only six students returned this final survey sent to all MSc students. The results are shown in Table 9. Three were participants and also returned their participant survey form, the other three were non-participants. From other case study data we know this was not a representative sample, the non-participants included one outlier case (Case 0625, discussed in more depth later in this thesis), and the participants included one
case with a largely non-programming project (Case 0506) for which many of the SERE recommendations were not relevant. Even if these exceptions were ignored, three of each case is just too few to draw a comparison between the participants and non-participants, though the overall adoption of good software engineering practice is higher in this sample.

<table>
<thead>
<tr>
<th>Others code</th>
<th>SERE participant</th>
<th>Easy to use your code</th>
<th>Journal</th>
<th>high level design</th>
<th>Source control</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No Case 0625</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes Case 0506</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes Case 0629</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No Case 0617</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes Case 0628</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No (No case details)</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 9 Final MSc Survey responses from all students 2005-2006

Students in this cycle also typically did not give yes or no answers to the questions, but rather gave extended open responses. This has led to a significant amount of interpretation on some of these results. For example in the use of source control, all of the participants clearly answered “no”, yet all of them also provided details of their manual methods or IDE support that provided limited protection against data loss. Given the capacity to recover from data loss was the purpose of the question, these are all positive results. The full responses students provided are presented in the appendix (pg 295). Other information was gathered from open questions on reuse, problems encountered and the use of software engineering. These are now presented.

**Facilitating Reuse**

Asked about the ease of using their code a student responded, “some is really straightforward and well documented, with assumptions and problems and even the design discussed in the documentation. On the other hand, some of the feature... [are]
complex”. This mixed response shows that at least an attempt was made to facilitate reuse. This was not the case for all students as became obvious in the responses to a question about deficiencies for reuse.

Asked for one thing to make it easier for others to use their code, one non-participant noted that their code “does not entirely match the [language] standard”. Another student suggested it would be useful for them to “Implement the ‘future work’ for each component”. The final non-participant suggested they could “comment it”. One of the participants also suggested “more comments”, though in this case they meant additional comments where as in the case of the non-participants the code was totally uncommented (as later interviews revealed). Other participants suggested “lots of high-level diagrams explaining how and why things exist in the way they do. This could be textual, but I think visually would really help people focus in” and that they could “simplify the [features named]… [but are] not sure how”.

**Problem encountered**
When asked about problems they encountered one respondent replied: “Procrastination and laziness”, another noted that they had trouble “Keeping track of referencing”, a third commented on communication issues with supervisors. One student comments that they “did not understand what was required for me, e.g., in terms of the report or the nature of the project” they listed “ask Andre, ask supervisor, search web” as possible solutions. The “steep learning curve” of their chosen development environment was noted by one student, and mentioned a work-around they had to use to make their programming language / environment compatible with dOxygen. One student (a non-participant) noted that “the only real problem was with poor code. It was developed in a component based manner so I'd finish one component before beginning another. This meant that if a component was faulty it would affect later components. Unfortunately this did happen on a few occasions so it meant recoding previous components. I suppose it could be put down to a poor development plan”.

**Use of software engineering**
When asked “If you had to do it again, would you use more or less software engineering?” responses generally responded that the level of engineering was about right or could be increased.
• “Maybe a little more, just in the realms of developing the components in a different manner”
• “About the same”
• “same... I reckon the balance was right [though] if you *need* a more/less answer, - more: - I'd have done the process model sooner”
• “If I had a decent UML tool that would actually support [environment], then yes. Unfortunately, tool-support is non-existent for this environment”.

Two students found the question ambiguous “I am not sure what you refer to software engineering here, but I would probably do things about the same. If you are referring to anything that isn't just programming, then I suppose it would depend on the feedback that got from my supervisor on how much documentation counted to the overall mark – some documentation helped me produce my tool, but some was superfluous outside of being awarded a mark”. The other student replied, “that's a ridiculously vague question. (Given that the job title ‘software engineer’ often means ‘programmer’, the question could be reduced to ‘would you code more less’, among other possible interpretations.)”. While the first of these answers (from a participant) shows a cost benefit analysis in their consideration, the second is an outright rejection of software engineering, or perhaps a lack of understanding. The initial survey showed quite clearly that most students saw software engineering as more than just coding.

A more general question on the role of software engineering in research was also asked. This questioned had already been asked in the initial survey, though student now had their own experience to draw upon.

The answers (see appendix pg 296 for details) highlighted a difference of perspective between the participants and non-participants. Participants are concerned with flexible methods (e.g. frameworks) and being able to tailor software engineering to their needs. Reuse is mentioned as a benefit for projects of all sizes and the underlying value appears to be in quality of product and process. This contrasts to the non-participants who focus on efficiency, tools, and promotion of the ad-hoc approach. These two groups are likely to be at different stages in our CARE framework.
Participation
The three non-participants were also asked whether with hindsight they would have signed up. All three responded negatively (details in appendix pg 297).

The concerns related to the potential for extra work, their lack of need for assistance, and the nature of the project itself. The underlying theme however (from these answers and other data) seems to be a desire to be independent and not have others become involved in their work. This is the approach of the artist rather than that of the scientist. Such approaches may produce exceptional results on occasion, but are unlikely to do so consistently. By regarding the entire process as a creative endeavour (rather than isolating the repetitive tasks from the creative ones) people with this approach render their process outside the scope of process improvement.

7.3 The Final Survey – Participants
The final participants survey sought to gather information on students perception of SERE and of the components within SERE. In addition to open questions, one critical aspect of this survey was the utility rating students applied to the various SERE components.

Students were asked to rate the components on their usefulness (to participants). The scale was given as: 1=useless, 2=some use, 3=useful, 4=very useful, 5= the most useful bit. The results were then scaled so each student contributed the same amount over all, divided between the components based on the students’ ratings. The final result is a ranked order for the process descriptors. The full working for the first year is provided in the appendix (pg 298).

Other questions in the participants survey asked participants for their impression on the costs and benefits of SERE participation, and their thoughts for improving SERE. Students overall impressions of SERE were also collected, and the survey included an opportunity for students to give advice to future students thinking of signing up. A selection of these quotes were used when introducing SERE to the second and third cycle students. The participant survey provided not just analysis, but also feedback in the ongoing evolution of the SERE approach. As participants were more involved, the level of detail was far greater than in the general post project survey previously discussed. For clarity the results in this section are again present in their cycles.
7.3.1 The First Cycle
In the first cycle nine students returned a participants form. The results in the first cycle can be divided into two parts, the usefulness of the process descriptors provided (as ranked by the students in a survey), and the reusability of the students projects in the opinion of both the students and the supervisors. The second set of data arises out of our intention to reset some of the projects then allow them to be reused by future students. This aspect of the experiment was later determined to be infeasible, however the data collected adds an additional perspective on reusability.

7.3.1.1 Usefulness of Process Descriptors
The full working for arriving at Table 10 is shown in the appendix (pg 298). The table shows that journal advice and the getting started guide were the most useful tools, while the website and the coding guidelines were seen as the least useful. The data is presented in the order of the original questions. The values have been calculated based on the first five tools, with those containing missing data elements (i.e. data from the control group who were not given access to the final two tools) being calculated separately.

<table>
<thead>
<tr>
<th>Process Descriptor</th>
<th>0.83</th>
<th>1.25</th>
<th>0.71</th>
<th>1.67</th>
<th>0.71</th>
<th>0.77</th>
<th>1.07</th>
<th>0.91</th>
<th>1.82</th>
<th>% of weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting started guide and general guidelines</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15%</td>
</tr>
<tr>
<td>Journal advice</td>
<td>0.42</td>
<td>0.83</td>
<td>1.43</td>
<td>0.83</td>
<td>0.71</td>
<td>1.92</td>
<td>1.79</td>
<td>1.36</td>
<td>0.91</td>
<td>16%</td>
</tr>
<tr>
<td>Tools webpage</td>
<td>0.83</td>
<td>0.83</td>
<td>0.36</td>
<td>0.83</td>
<td>1.07</td>
<td>0.77</td>
<td>0.36</td>
<td>1.82</td>
<td>0.45</td>
<td>12%</td>
</tr>
<tr>
<td>CVS</td>
<td>1.67</td>
<td>0.83</td>
<td>1.43</td>
<td>0.83</td>
<td>1.43</td>
<td>1.15</td>
<td>0.71</td>
<td>0.45</td>
<td>0.91</td>
<td>15%</td>
</tr>
<tr>
<td>Coding guidelines</td>
<td>1.25</td>
<td>1.25</td>
<td>1.07</td>
<td>0.83</td>
<td>1.07</td>
<td>0.38</td>
<td>1.07</td>
<td>0.45</td>
<td>0.91</td>
<td>13%</td>
</tr>
<tr>
<td>Joint meeting with supervisor</td>
<td>0.83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14%</td>
</tr>
<tr>
<td>Technical review</td>
<td>1.25</td>
<td>0.83</td>
<td>1.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14%</td>
</tr>
</tbody>
</table>

Table 10 Relative usefulness of process descriptors 2003-2004
7.3.1.2 Reusability of Projects
The survey conducted with supervisors (appendix, pg 246) and results from the students survey’s reuse questions were compared. The results in Table 11 compare the student’s opinion and the supervisors opinion on whether the code could be reused (first two columns after the ID, coloured black for positive answers) followed by a rating for how likely to it to be reused (10 being certainty of reuse, and 1 being certainty it cannot be reused) . The final column indicates is supervisors felt the project would be useful for a project in Cycle Two.

<table>
<thead>
<tr>
<th>ID</th>
<th>Student reset</th>
<th>Supervisor reset</th>
<th>Student score</th>
<th>Supervisor score</th>
<th>Next year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>Yes</td>
<td>4</td>
<td>8</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td>7</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td>7</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>Missing</td>
<td>1</td>
<td>Missing</td>
<td>Missing</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>Missing</td>
<td>5</td>
<td>Missing</td>
<td>Missing</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Missing</td>
<td>6</td>
<td>Missing</td>
<td>Missing</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Yes</td>
<td>3</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>No</td>
<td>5</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>Missing</td>
<td>No</td>
<td>Missing</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>11</td>
<td>Missing</td>
<td>No</td>
<td>Missing</td>
<td>2</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 11 Student and supervisors views on resetting projects 2003-2004
Examining the data in more detail, only two cases (two and three) received a vote for resetting from both the student and the supervisor. These cases were from different supervisors.

A more in-depth look at Case 1 shows that the student’s reason for not wanting it reset (despite having a project their supervisor was very keen to reuse) is that they didn’t wish to be hassled by future researchers. This is discussed later in section 9.8.2 (pg 211). One member of the experimental group (subject 5 in the tables above) provided a very interesting extended response on the question of benefit. This will be discussed in section 9.4 (pg 200).

As can be seen there was very low potential for projects being able to be continued in the following year, and even if more project were capable of being extended the following year, it would be up the new students to decide if any of them wished to take on those projects. This information was one of the factors that made the original
experimental design (including the resetting of projects and their reuse) unfeasible in the time available for this research.

### 7.3.2 The Second Cycle

In the second cycle seven participants returned final survey forms. The results are presented here in three sections. The first examines students reflections on the usefulness of the process descriptors. The second discusses the cost and benefit of SERE participation. The final section discussed the impact and effectiveness of SERE.

#### 7.3.2.1 Usefulness of Process Descriptors

In Table 12 we apply the same approach as used in the first year to evaluate the weighted relative usefulness of the various process descriptors. The final column provides the utility ranking and the penultimate column provides the score this is based on. The score has been expressed as the percentage showing the utility given to that item as a percent of the total utility spread across all tools.

<table>
<thead>
<tr>
<th>Process Descriptor</th>
<th>% of weight</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3a. Initial lecture</td>
<td>0.07</td>
<td>6</td>
</tr>
<tr>
<td>3b. Getting started guide and general guidelines</td>
<td>0.09</td>
<td>1</td>
</tr>
<tr>
<td>3c. journal advice + sample journal</td>
<td>0.07</td>
<td>7</td>
</tr>
<tr>
<td>3d. tools web page</td>
<td>0.09</td>
<td>2</td>
</tr>
<tr>
<td>3e. CVS</td>
<td>0.07</td>
<td>14</td>
</tr>
<tr>
<td>3f. Allways Sync</td>
<td>0.07</td>
<td>11</td>
</tr>
<tr>
<td>3g. Dia</td>
<td>0.04</td>
<td>16</td>
</tr>
<tr>
<td>3h. GIMP</td>
<td>0.04</td>
<td>15</td>
</tr>
<tr>
<td>3i. dOxygen</td>
<td>0.09</td>
<td>8</td>
</tr>
<tr>
<td>3j. coding guidelines</td>
<td>0.04</td>
<td>3</td>
</tr>
<tr>
<td>3k. Initial meeting with me (April / May)</td>
<td>0.04</td>
<td>4</td>
</tr>
<tr>
<td>3l. Technical review meeting (July / August)</td>
<td>0.07</td>
<td>12</td>
</tr>
<tr>
<td>3m. Other meetings and e-mails with/from me</td>
<td>0.04</td>
<td>9</td>
</tr>
<tr>
<td>3n. dOxygen sample code and diagrams</td>
<td>0.07</td>
<td>5</td>
</tr>
<tr>
<td>3o. dOxygen installation and setup instructions</td>
<td>0.07</td>
<td>10</td>
</tr>
<tr>
<td>3p. Technical review preparation document</td>
<td>0.04</td>
<td>13</td>
</tr>
</tbody>
</table>

The order of perceived usefulness this year was: Getting started guide and general guidelines, tools web page, coding guidelines, Initial meeting (April / May), dOxygen
sample code and diagrams, Initial lecture, journal advise + sample journal, dOxygen, Other meetings and e-mails, dOxygen installation and setup instructions, Allways Sync, Technical review meeting (July / August), Technical review preparation document, CVS, GIMP, and Dia.

### 7.3.2.2 Cost and Benefit of SERE Participation

**Time**

Three students said participation took about the time they expected it would. Three said it took less time than expected. The final student noted that they had limited participation due to perceived cost.

**Systematic approach**

SERE also improved the systematic nature of the approach, while keeping the burden manageable. Six students said participation encouraged them to do more software engineering than they would otherwise have done. One of these commented “I've realized how it helps, as the project grows, to keep a clear vision of it”. One student said it encouraged less software engineering and added that “from the beginning you specified the differences found in research code”. This student had industry experience prior to their MSc and it is possible they would have over engineered without SERE’s recommendations.

**Benefits**

Comments in the final survey that related to how SERE participation helped the students included: “It did only for the website (I used dOxygen generated html)”, “YES, because I used the code comments and the diary as a rough version of what I wrote in the final report”, “gave me another opinion other than supervisors”, “yes, obviously the advice on keeping a journal which reinforced what I already new is very important – this is a must do. Help with analysing at what stage the project is at, how much more is to be done and having a further opinion on what has been done so far and what is still to be done”, “Would have helped if I could use dOxygen with my code”, “Yes, having a third persons opinion is very helpful”, “The documentation part can be used in any kind of project. Also, for starting a project from scratch, the engineering part is also very helpful”, “Yes, always having another point of view is very helpful. Also helped for organizing the work.”
When asked if they benefited, six said yes and one said they had not looked at the website. When asked how they benefited responses were “Had the chance of getting some extra help, DOxygen is very useful”, “another person to discuss the project with. Continual monitoring of progress in form of what you should be doing or have done. The review I expect would put into perspective the project – something which is hard to see when you are stuck right into it and do not come up to see the overview of it”, “by knowing about some tools which can be used for the project” and “Yes, always having another point of view is very helpful. Also helped for organizing the work”.

7.3.2.3 Impact and Effectiveness of SERE

When asked about the impact of SERE Guidelines and given the choices: A didn’t read, B none, C some, D a lot, two respondents said they had not read SERE guidelines, three said it had some impact, and two said it had a lot of impact. When asked whether (with hindsight) they would follow the guidelines more closely or less closely, two students replied “yes” (not a very useful answer), two replied “the same”, and two replied “more” with one of these qualifying it as “a little more”. No-one said they would follow the guidelines less. The lack of any coercion to use the guidelines means any students who felt the burden was too high would likely have reduced their involvement at that time. It also shows that the guidelines were of benefit even when not fully implemented.

When asked for their overall impression of SERE students were very positive. The replies (in addition to non-responses) were:

- “Good. Could be better if you weren't away for the long period you were” (in reference to two weeks I was away at a conference)
- Two students replied “good”, one of whom appeared not to have used anything, but commented that the concept is a good idea
- “Very good”
- “More hands-on view of how SERE works, the first lecture did not show all the potential, which could incite more interest in participating”

When asked what advice they would give next year’s students on getting involved, responses were:

- “Do so, and learn all you can about what to do, before the dissertation”
• “it’s totally worth [it], and can help to understand the value of software engineering”
• “do – it’s another line of support”
• “Do it!”
• “Participate in groups and discuss with peers about using SERE”.

The last comment came from an international student (non-native-speaker) a group interview examined this issue of groups in more depth and also highlighted that language was a barrier, any extra reading in English that was not a requirement was seen as a high barrier to this group of students. Overall though, again very positive feedback that suggests SERE was appreciated by students and adoption would be encouraged by former participants.

Students were also asked how well SERE met its goal of "maximum benefit, with minimum effort, now" responses included: “quite well”, “students are chased up to be given support usually it would be the other way round. The tools you would be well advised to use are assembled for you. Advice on coding, backup, versioning issues etc is given without first having to ask the question”, “It indeed does. I thought it would need more time from my part.” This is a positive reflection on SERE meeting its aims.

7.3.3 Cycle Three

Only three participants returned final survey forms in the third cycle. The results are again presented in sections this time the section are; process descriptor usefulness, cost and benefit of participation, impact and effectiveness of SERE, long term benefits, and future developments. The final two section were a result of additional questions asked this year as the experiment was drawing to a close.

7.3.3.1 Usefulness of Process Descriptors

The result for process descriptor preferences are shown in Table 13, in the same format to that used for the second cycle. The initial rankings were again 1=useless, 2=some use, 3=useful, 4=very useful, 5=the most useful bit. As can be seen in subject one in Table 13, there was also the option to not rate a tool… and this subject was only willing to express an opinion on a small selections of the tools. This is likely a result of them not knowing enough about the others to express an opinion. This is likely due to a low level of involvement from the student.
Table 13 Relative usefulness of process descriptors 2005-2006

<table>
<thead>
<tr>
<th>Process Descriptor</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>% of weight</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>3a. Initial lecture</td>
<td>0.23</td>
<td>0.02</td>
<td>0.11</td>
<td>11.9%</td>
<td>2</td>
</tr>
<tr>
<td>3b. Getting started guide and general guidelines</td>
<td>0.15</td>
<td>0.09</td>
<td>0.05</td>
<td>9.8%</td>
<td>5</td>
</tr>
<tr>
<td>3c. journal advice + sample journal</td>
<td>0.08</td>
<td>0.04</td>
<td>0.05</td>
<td>5.8%</td>
<td>8</td>
</tr>
<tr>
<td>3d. tools web page</td>
<td>0.15</td>
<td>0.11</td>
<td>0.05</td>
<td>10.6%</td>
<td>4</td>
</tr>
<tr>
<td>3e. CVS</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
<td>1.6%</td>
<td>14</td>
</tr>
<tr>
<td>3f. Allways Sync</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
<td>1.6%</td>
<td>15</td>
</tr>
<tr>
<td>3g. Dia</td>
<td>0.00</td>
<td>0.09</td>
<td>0.03</td>
<td>3.8%</td>
<td>10</td>
</tr>
<tr>
<td>3h. GIMP</td>
<td>0.00</td>
<td>0.07</td>
<td>0.03</td>
<td>3.1%</td>
<td>13</td>
</tr>
<tr>
<td>3i. dOxygen</td>
<td>0.15</td>
<td>0.11</td>
<td>0.08</td>
<td>11.5%</td>
<td>3</td>
</tr>
<tr>
<td>3j. coding guidelines</td>
<td>0.00</td>
<td>0.11</td>
<td>0.08</td>
<td>6.3%</td>
<td>7</td>
</tr>
<tr>
<td>3k. Initial meeting with me (April / May)</td>
<td>0.23</td>
<td>0.09</td>
<td>0.13</td>
<td>15.0%</td>
<td>1</td>
</tr>
<tr>
<td>3l. Technical review meeting (July / August)</td>
<td>0.00</td>
<td>0.11</td>
<td>0.11</td>
<td>7.2%</td>
<td>6</td>
</tr>
<tr>
<td>3m. Other meetings and e-mails with/from me</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.0%</td>
<td>16</td>
</tr>
<tr>
<td>3n. dOxygen sample code and diagrams</td>
<td>0.00</td>
<td>0.11</td>
<td>0.03</td>
<td>4.6%</td>
<td>9</td>
</tr>
<tr>
<td>3o. dOxygen installation and setup instructions</td>
<td>0.00</td>
<td>0.00</td>
<td>0.11</td>
<td>3.5%</td>
<td>11</td>
</tr>
<tr>
<td>3p. Technical review preparation document</td>
<td>0.00</td>
<td>0.00</td>
<td>0.11</td>
<td>3.5%</td>
<td>12</td>
</tr>
</tbody>
</table>

The order of perceived usefulness this year was: Initial meeting (April / May), Initial lecture, dOxygen, tools web page, Getting started guide and general guidelines, Technical review meeting (July / August), coding guidelines, journal advise + sample journal, dOxygen sample code and diagrams, Dia, dOxygen installation and setup instructions, Technical review preparation document, GIMP, CVS, Allways Sync, and finally Other meetings and e-mails with/from me.

From other data collected from these students we can see that even the items further down this ranked list were of some use to the students. The results again show that different tools were perceived as most helpful by different people, however certain process descriptors were found to be much more valuable (in general) than others. The 15% rating for the initial meeting is a result partly of high rating given by the first subject, and partly due to the other two also rating the initial meeting very high. The conclusion would be that if only one process descriptor could be afford for all students, an initial meeting with a software engineer would be the best thing to do. While this sample is particularly small, the second cycle results agreed with the high utility of the initial meeting. Though it was ranked fourth in the second cycle, the difference between the top four items was not that great. On such small samples the exact order of the ranking is not particularly reliable, however, the relative proximity
to the start or end of the list does give some indication, specially when taking in conjunction with other data on these process descriptors.

### 7.3.3.2 Cost and Benefit of SERE Participation

#### Time
Two students said participation took more time than they expected it would, and one said it took less. A new question introduced for the final cycle asked if more, the same, or less contact time would have been better. One said “the same”, the other two said more and elaborated that they would have preferred more group time with other students. One of those wanting more contact time also clarified “I think that in the future I would want less as I would now know a lot of the stuff...but if I went back 4 months, it would be MORE”.

#### Systematic approach
When asked if SERE encourages more or less engineering than they would otherwise have done, all three answered “more”. One made specific reference to the process document as an example of something they otherwise wouldn’t have done. The same student noted “It is difficult to quantify the amount I would have done, probably in a different form, without SERE, but I suspect with SERE it is more. For example, some of the stuff identified by the first technical review”. Another was more emphatic; “More, without a shadow of a doubt”.

#### Benefits
When asked if they felt they benefited from being involved the responses were all positive. Specifically students said: “yes, learning new things, increasing some understanding about parts of the project (e.g., nature of research), forcing me to keep up with certain chores (e.g., tidying up), encouragement, etc”, “re-enforced the need for good engineering practice during implementation phase particularly forced much better code documentation to be kept process model was a good reflective organisation tool all of above helped in write-up”, “Yes good to talk”.

Comments on how SERE helped the students were: “it forced me to organise all of the completed tasks and associated information into a logical manner, which made structuring the report a LOT easier the implementation section was considerably easier to write due to the extended code commenting”, and “yes, but not much. It
helped by the way of the dOxygen generated class diagrams in terms of clarifying relationships, though this wasted time in trying to use the diagrams from DOxygen due to problems with them (e.g., inaccuracy). Some of the comments, whilst I felt little need to copy verbatim, were useful in helping understanding some code, esp. why we need something and what it does. Looking back over the process document helped me discovery issues and ideas for my report.” The third respondent was the one with limited coding and did not provide a response to this question.

**Reuse**

One respondent rated their codes potential for reuse a nine, another rated it a three and the third comments that it was “a starting point” for further research. The respondent who rated the potential for reuse a nine when questioned on resetting their project responded “although it would be possible to convert the project into a commercial product I feel that it would be more rewarding to see if the principle… can be transferred elsewhere”. The respondent who rated their reuse potential a three responded “Yes, but only if it was to be reused properly. There are clearly bugs, outstanding synchronisation issues, incomplete / inaccurate documentation.”

### 7.3.3.3 Impact and Effectiveness of SERE

In commenting on the impact of SERE guidelines all the students indicated that it has “some impact”. The students whose project involved minimal coding qualified this as “a little impact”.

When asked if they would follow them more with hindsight two students said “more”, and one said “the same, maybe more”. The student who said they would follow them the same or possibly more had taken part in all stages of SERE. They also noted “a problem I have is that they aren't comprehensive or well-tailored to individual projects. I take the example of a book on MSc computing projects (of which there are several) - these may offer more of the kind of comprehensive advice I am looking for. It would have helped to have the supervisor there to help me interpret the guidelines for my particular project.” It should be noted that the meetings with supervisors were dropped after the first cycle. The closest was when a supervisor attended a technical review in this third cycle. More interaction between supervisors, students and the SERE software engineer would definitely be of benefit, but the logistics and need for
as much consistency as possible limited the amount that could be done within this experiment.

When asked what advice they’d give next year’s students the responses were positive but reflected the new ownership by students of their research process:

- “it’s not a magic wand, you’re only going to get out what you put in”
- “look at it early!!”
- “If you have problems with projects and are not completely sure or think you need someone to keep you on the right track, then it may help. Don't feel forced into everything or guilty if you don't do it all - tailor it to your own needs.”

Commenting on their overall impression of the running of SERE the students were again very positive, They commented:

- “I think the department needed to offer you a little more consideration, you seemed to spend much of your time fighting the department for our attention, where they should probably have set aside some time or at least organised some official venue for lectures/talks/reviews”.
- “It got better as it went on - e.g., promises about workshops went by unfulfilled, ... But things like the poster, and extra help outside of the reviews did help. As for the introduction, reasonably good but could have helped by providing some kind of leaflet.”
- “needs more ‘push’ from supervisors/lecturers”.

Students were also asked how well SERE met its goal of "maximum benefit, with minimum effort, now” responses included: “it does this relatively well.... but I think a better summary would be to ‘best utilise the most time-efficient engineering techniques available.’”, “Well, it certainly requires effort, and the results, at least as far as getting a higher mark are concerned, is questionable given the amount of work. In reality, the amount of work was, generally, minimised; but the work that was required was, at least generally, non-invasive or obstructing. i.e., they either fulfilled other necessary parts of the project as a by product or consumed 'spare' time. In summary, for the goal of getting a higher mark, I see the SERE as I participated in as not generating either significant confounding effort or beneficial results.”, the final
respondent who had the largely non-coding project responded “not for my project”. From this we can see that SERE really is a software engineering approach, not simply a general research improvement approach. Further, while one student doubted the benefit in marks, the benefit to the quality of the project was commonly accepted and our data on student marks does additionally show an improvement. Taking these two ideas together we observe that SERE does not involve “game playing” to increase marks but rather focuses on actually quality improvement. It is a positive reflection on the marking that such improvement in the product’s quality and in the process quality does actually seem to lead to improved marks, regardless of who the examiners or students were. Again it should be stressed that in this experiment the marking and the experiment were completely independent.

7.3.3.4 Long Term Benefits
Given this was the last cycle, students were asked if SERE should be incorporated into the MSc course as a standard component of it. The responses were:

- “yes, but modified. I think is only as useful as students are willing to make of it. Convincing them or bribing them may help influence them to take it up. But I fear that without the support of a dedicated individual, such as Andre, it will go stale (out of date, no innovation), be seen as a chore, and be a general disaster.”
- Another student joked “I think you should be allowed to run the experiment for another year, paid of course ;)” then added, “All joking aside, yes it should: - we've had a slight edge because of it this year and the department would be stupid not to offer assistance that could earn people a few extra marks.”
- The final respondent commented that it “should be included”.

These were all very positive comments, the first comment about the need for continual improvement matches the needs of the final stage in the CARE framework (pg 63). It may be taken as an indication that the experiment on a meta level did move the departments capacity up to this level, but as the student notes, only for those willing to take advantage.

Students were also asked if they would use any aspects of RAISER for other projects and which parts, and if they would have to be research projects. Two students replied:
• “I’m converted to the code commenting guidelines and doxygen as a program.... and no, the skills should be transferable”

• The other student replied:
  o “dOxygen - sure, generally it is better than Javadoc”
  o “process document - maybe, but in a longer form of research (e.g., a PhD), it may be a bit too much and I could use alternative forms”
  o “technical reviews / advise sessions - sure, these are of some use and even if they don't bring up any critical issues, they provide reassurance that you are on the right track. I suppose it also matters how much trust you have in them.”
  o “Advise - probably not since it is a little too specific (though I would argue not enough) to the MSc / short projects”
  o “Only research projects? Well I don't know. The only project I can delineate are: personal coding projects and work-based projects. In the former, probably not though I should, due to the informal nature of the work I cannot switch on the mindset of doing things properly. As for the work, it would depend on the environment - if I was offered a choice and nothing better was offered, then of coarse I would , and I would likely use them more than with my MSc since things like reuse and collaboration are much more important”.

The students responses show long term adoption of some SERE process descriptors, the final comment suggests the benefits for reuse and collaboration were apparent to the student but not deemed important in the context of the MSc project. This view is supported by other evidence already discussed where the student makes a distinction between product or process improvement and increased marks.

7.3.3.5 Further Developments

When asked what tools they found most useful one student responded, “dOxygen, dOxygen and dOxygen ;) -- I already knew of Dia/Gimp and endnote was moderately useful”. Another replied “dOxygen wizard” followed by a list of 16 additional programs not included in SERE. The list ranged from openoffice through to Firefox with a Google toolbar. This open question shows there is still plenty of scope for
additional process descriptors related to generic as well as more research topic specific tools.

When asked what else would have been included one student replied “personalisation, comprehensiveness, more help with the report, justification for approach, a Process Document that was clearer and better able to be integrated, full versions of software, printed documents to save out printing budget, more tools in more categories with relative merits, hand outs, fill in sheets to chart progress and ensure we didn't miss things, a FAQ, guidelines on timing”. When asked what they’d like to see changed, they added, “more context specific”, “more more more! (but not increasing the burden, just increasing the choice)”, “remove bias towards opensource / free software and include proprietary, esp., when it is available for free through the department” (e.g., Visio, One Note), “some sort of glossary that helps clarify what terms are”, “customise the process document to different languages / Java versions”, “add some kind of generator tool that is able to tailor stuff to the needs of a person / project - e.g., do you like to do lots of documentation or not, do you respond to visual or textual?”, “add some kind of online progress chart - though may be unused”. These are all very details suggestions but provide good suggestions for future work in this area.

Another student suggested “some technique for better planning/executing research... I was happy with my own recording technique after the fact, but I feel had my research phase been better planned and organised (unsure how) I may have been able to squeeze it into a little less time” this student also suggested SERE could be improved through additional information: “the website is a little bare, more supporting/background information would make SERE more accessible”. Again these responses suggest ways that SERE could be made more complete and professional if additional resources (such as Software Development Lab) were made available.

### 7.4 Six Questions in Cycle Three

All students were asked six questions (see pg 241) as they were completing their projects. Eight students replied. The questions focussed on problems students had faced, areas where they wasted time, useful tools and advice they received, their use of diagrams, their approach to the research, and their wish-list to make their research
process easier / better. The student feedback used to compile the lists in this section may be seen in the appendix (starting at pg 300)

### 7.4.1 Problems and Wasted Time

The problems students faced can be broken down into two types, essential difficulties and accidental ones. These terms are from Brooks (1986) and have been introduced previously (see pg 11). They form a critical element of this work which recognises it is only the accidental difficulties that can be targeted for improvement. Essential difficulties are part and parcel of doing research while accidental ones are perhaps also accepted as “normal” when they are in fact “wasted time” that should be eliminated from the process, or at least mitigated as much as possible.

The Essential difficulties:

- Defining the task
- Deciding a course of action to tackle the research problem
- Dead ends in the research (ideas that on evaluation don’t work)
- Managing the complexity of the problem and solution space
- Understanding literature, research approaches and new terminology
- Engineering difficulty in building the solution and understanding the tools (and programming languages) available to do so

These examples all reflect the inherent nature of the task, the essential task as explained by Brooks (see pg 11). Such challenges are to be expected and cannot be removed from the research process.

The Accidental difficulties:

- Repeated failure in one area of the code (this suggests a design flaw)
- Poor time management
- Over design or too rigid design for the research environment
- Resource difficulties
- Communication difficulties
- A lack of project management / poor project management generally leading to bottlenecks in the process
Chapter 8  Evaluating Components of SERE

The accidental difficulties relate to wasted effort and time pressures. Wasted effort as previously explained (pg 6), is avoidable effort often spent on non-creative tasks. SERE aims to reduce this type of effort where possible, though other forms of mitigation may also be needed to improve efficiency, reduce frustration, and allow research to spend their time more productively. The SERE components aim to assist with this.

7.4.2 Assistance Received and Tools Used
Responses indicated that participants received more advice and used feedback opportunities in a more systematic manner than non-participants did. For participants doxygen was warmly embraced and for many became the focus of their response to questions about tools.

Students were also questioned about their use of diagrams. Two non participants did not use diagrams at all, another produced designs for components but no over-all architecture diagram. These responses are in contrast to the participants where all but one created an architectural diagram, and that research project was an exception with very little coding.

7.4.3 Approach to the Project
A great deal of detail was collected on students’ approaches to their projects. The data shows a great deal of variety, but with participants taking a more systematic approach and also commenting on aspects of SERE that they integrated. Problems of both a technical and human nature impacted on students’ processes. The need for agile approach was again highlighted. One student completed their project while on a placement in industry. The experience they describe differed markedly from other projects, the key aspect being the amount of coding and the high level of supervision and technical advice available.

7.4.4 Making the Research Process Easier
Both participants and non-participants were concerned about official guidance and a more systematic structure to work within. The lack of structure could itself be seen as allowing students and supervisors more freedom, but there is a difference between stating there are no formal requirements on presentation or structure and not providing
any guidance. This is an administrative matter and outside the scope of this dissertation.

7.5 The Case Studies
For a thematic discussion of the issues presented in the cases please see chapter 8, and for a discussion of the various cases outcomes and opinions on SERE please see chapter 9. The individual cases presented here will refer with page number to places where further comments from this students can be found, and the comments used later will reference the case number to allow the reader to refer back.

Table 14 presents the cases individually referred to in this thesis. Data from other cases has been included in statistical analysis and in survey response data, but in these instances it has not been appropriate to identify any students by case. The first two digits of a case code represent the completion year of the cycle, i.e. 04XX would represent a case in the first cycle, 05XX one in the second cycle and 06XX would represent a case in the third cycle. Case numbers have been used to protect the identity of the participants, however, knowledge of the cycle they participated in is intentionally provided to set them in content. In Table 14 the group codes are ‘P’ for participant with an indication of ‘(E)’ to denote experimental group members and ‘(C)’ to denote control group members, and ‘N’ to denote non-participants. Further tables with details on the cases in terms of benefits and other influencing factors can be found in the appendix (pg 310).
Table 14 Cases referred to in this thesis

Further detail about the case study sample each year and a selection from the above set of case studies will be presented in this section in more details. These cases have been chosen to highlight the differences between the cases and provide additional detail on those cases mentioned most regularly in this thesis.

7.5.1 First Cycle Case Studies

Of 30 MSc students 16 volunteered to participate. Of the participants, eight of these were assigned to the control group. The selection between control group and experimental group was random (numbered off according to the pattern Control, Experimental, Experimental, Control). An expert (the MSc administrator) was consulted and did not see any pattern in the selection of those that volunteered, nor in
the selection between control and experimental groups. Patterns looked for included nationality, students who did undergraduate study at the same institution c.f. elsewhere, promptness of submission of coursework and provisional grades for coursework.

7.5.1.1 Research Case Study 0405
This student was a participant assigned to the experimental group. Their work involved reusing an open-source project largely developed by PhD students. The code was largely undocumented except at the abstract level and this made the task difficult (see pg 211). Asked after the project what one thing they could do to make their project more reusable the student replied “Complete the code comments”. They explained that although they did use dOxygen for the documentation as requested (and found the coding Guidelines helpful) their documentation tailed off towards the end under time pressures (see pg 171). The student felt the benefits of SERE, and of documentation in particular, would largely be for follow-on research rather than the initial research like themselves, however they did note that the documentation effort helped them in their understanding (see pg 200).

Observations:
This case study shows themes that would reoccur throughout the three years of the experiment, as well as the starting point of minor benefits from which SERE improved to give more substantial benefits for lower cost as new process descriptors were added. Reoccurring themes in students’ comments included the discussion of benefits for them versus benefits for future developers, the idea of time pressures and time becoming more “expensive” later in the project, and use of documentation both to share knowledge and to focus their own thoughts.

7.5.1.2 Researcher Case Study 0404
In this instance a student in the control group had the task of extending an existing piece of code. The student found it difficult to understand code and documentation that had been put together by other students, including a PhD student. Overall they felt they were able to do “much more” than if they had started from scratch. At the same time, however, we can see from their responses that they spent two weeks learning about the code, two weeks unsuccessfully trying to solve dependency issues after a hardware migration (solution was to migrate back), and one week (three to four days)
trying to merge two existing programs before resorting to calling one from the other. This suggested that the reuse was not a smooth process and had room for improvement.

The student concluded that the biggest problem for someone following them is that it would take “time to understand [the code] because there is a lot of files and code” they also concluded that if they could change one thing, it would be to remove functions and make the design simpler.

Despite being in the control group, the student found the journal advice and recommendation about version control “very useful”. Advice about coding guidelines was also found to be “useful”. The student felt they might use all these things in future projects.

Observations
This appears to be a classic case of getting a research student to undertake re-engineering while attempting to also do research. The student wasted a large amount of time undertaking mundane engineering tasks. The student is not an expert and has other priorities, so two ideas to improve the engineering of the product fail. In fact, the tasks were completed, but in a way that left the product only able to run on one machine and architecturally uses coupling that restricts future reuse of the code. In the end the student has wasted five weeks of project time. Had this project been reset beforehand this learning curve could probably have been reduced to at most a couple of days. For a short term project like an MSc, five weeks is a very long time to waste. While reuse has helped the student, it has done so in a highly inefficient manner.

7.5.1.3 Researcher Case Study 0420
The case involves a student with whom we have no contact beyond their final survey. They opted not to be involved in the project, knew little about the experiment except that participants had to comment their code, and were unsure if they would be involved given the choice again. This student would like to have used more software engineering, but couldn’t due to time constraints. Their project required a large background knowledge and their software was developed on an unstable (buggy) platform using many hacks. They stated that they did use high level diagrams, describing these as “box diagrams!” When asked what impression they got of SERE
from the other students they replied “they had to comment their code……that’s all I know!” The student noted that their own code was not commented.

**Observations**
This is a story of systemic faults. If the computing platform being used was stable and reliable, many projects including this one would become much more efficient to develop. A stable platform is essential to ensure the product lasts at least until the end of the project, and hopefully beyond. The student’s lack of documentation exacerbates this problem for anyone wanting to continue their work. This looks like a situation where a RESET effort could be of great value.

**7.5.1.4 Researcher Case Study 0418**
In this case the student was a member of the experimental group. The project used a number of open source tools. The student spent a lot of time learning and installing open source packages correctly, some of which were not used. The student felt the use of other packages saved them from having to learning to implement certain functionality themselves. This worked until local environmental variables interfered and the code needed to be adapted. The adaptation was successfully completed.

This student did not go through the coding standard prior to their review. In other ways they did make a decent attempt at SERE participation and discussed their project informally on a regular basis. Their FTR (like all the FTRs) occurred rather late in the process. They had only made a very small effort at improving the coding prior to the review (a few hours’ work immediately before submitting for review). Although they didn’t use dOxygen themselves in the end, for their FTR they were shown a version of their code marked up for dOxygen with examples of various styles and categories of comments. The technical issue of constructing comments in a format readable by dOxygen (and people) was easily understood during the FTR. Understanding of what to comment and why, and recognising these comments took much greater effort. In the one hour FTR comment types were understood and appreciated, although a significant part of the session was taken up by this. A few mistakes in the code were spotted and raised to the producer’s attention (this was appreciated by them). The output of dOxygen was found to be very useful for checking on naming conventions for variables and classes, as well as for checking structure. Total preparation time for
this FTR was two hours. The FTR preparation version of the code and resulting documentation was later used for other projects as an example.

When asked if they would follow SERE guidelines more closely if they did the project over they replied, “I would follow more closely the guidelines that I felt were necessary to be undertaken. In some cases, I was already following the guidelines without realising.” They also commented that they “benefited from the suggestions given by the [SERE] project head... The code review was also useful for tidying up/commenting suggestions.”

The student felt the amount of time that the experiment took up was less than originally expected but was “about right” as it “enabled the participants to get on with their projects without too much interruption”. The student’s post project data shows conflicting evidence as they report that coding guidelines were not very useful for them, but elsewhere in the survey encourage their use by others and said they found the FTR very useful for commenting suggestions. The student did not feel their project would be useful for others and did not recommend that it be RESET.

**Observations**

This case shows a clear need to run the FTRs earlier in the process. The student’s view of the coding standard as useless seems more related to the fact that they didn’t read it than any other factor. The desire to produce an “API / JavaDoc” and to comment better as well as the usefulness of the review for things that were supposed to be addressed by the coding standard shows that the value of software engineering was appreciated, although time pressures prevented it being used properly. This is supported by the comment that tools were not used but after seeing them used by others this was regretted.

This case represents a typical experience in the experimental group in the 2003-2004 set of cases, with the exception that while appreciated, the coding standards were not seen as particularly relevant. The view that the project would not be reused had a large impact on the types of software engineering used and the relative value placed on many of the RAISER suggestions which were seen as something to be done if time permitted, but not of particular importance to the research.
7.5.2 Second Cycle Case Studies
The second cycle case studies in 2004-2005 saw more supporting tools and efforts made to reduce the barriers to adoption. A full description can be seen in section 6.2.2 (pg 102). In addition, meetings with students (including technical reviews) were all recorded.

7.5.2.1 Researcher Case Study 0526
This student was a participant with industry experience and an industry-related project that they intended to continue after their MSc. When asked in the introductory lecture why software engineering might be of benefit to research they replied “In that way we discover more about the problem as we design and specify the software”. In an initial meeting the student explained that their project would adapt an open source product (see pg 211).

This student took part in an FTR, the only person to undertake an FTR in the second cycle. The review looked at code, documentation and a bit of the draft thesis. The student commented positively on the review, noting that having an extra and more external perspective was itself of benefit (see 179). The technical review was observed by a final year PhD student. During the review the student asked if the reviewer liked the drawings in the draft thesis. The drawings were not generated and were fairly detailed and well put together “I spent a lot of time in photoshop... I think drawing give you more information” the student explained. While dOxygen had been used there was little evidence of this in the output. The student was asked during their technical review what they thought of SERE. They replied “it's nice… I haven't felt pressured from it. But I don't know if you are satisfied with the things I've given you.” This student was the most involved in the second cycle.

In the post project interview the student gave the following overall evaluation of SERE, “It helped, just the very basic advice, how to do the design,. how to keep track of the work done, also the tools. DOxygen came up pretty... I documented the program quite a lot”. The student mostly documented to enable reuse (see pg 211) and also said they used the ideas in the Coding Guidelines rather than strictly following the guidelines themselves (see pg 175). The student’s technical review noted a lack of documentation and the student said they were going to do that later. The post project
interview shows that the student finished about a week early and then undertook the
documentation cleanup task – as they said they would. The student explains, “actually
I did the documentation part after I finished the report, after I put some nice diagrams
I found back in the report. After I finished more or less the report I started 3 or 4 days
just documenting.” They clarified that this did in fact lead them to put a couple of
doxygen-generated diagrams in the report. They felt other diagrams were not needed
as the generated documentation was available on their website and the address was
provided on the reports cover.

In their final survey, when asked if participating has been helpful the student replied
“Yes, always having another point of view is very helpful. Also helped for organizing
the work.” Their overall impression of how SERE was run was “very good” and they
felt it could best be improved through additional meetings. Asked if they would use
any aspects of RAISER on other projects the student replied “The documentation part
can be used in any kind of project. Also, for starting a project from scratch, the
engineering part is also very helpful.” When asked if SERE lived up to the idea of
maximum immediate benefit for minimum effort the student replied “It indeed does. I
thought it would need more time from my part.” The student indicated they would
strongly recommend others participate in SERE.

**Observations**
This student was working on an existing project in a way that could be described as
adaptive maintenance. This brought together the problems of research and the more
general problems commonly associated with maintenance of a large body of code. The
student responded by using software engineering, taking advantage of the tools and
opportunities SERE made available, but limiting the investment in a way they felt
would have the most benefit to them. The student’s efforts at documentation were
mostly to assist others and behave in what they saw as a professional manner, they did
not see this being of benefit to themselves. SERE has a significant impact on this
student’s work practices, but not in quite the way it had been intended.

**7.5.2.2 Researcher Case Study 0522**
This student was a non-participant with industry experience and an industry-based
project that took place outside the UK. In their initial survey the student felt that
Software Engineering was not useful to the researchers doing the initial research, but
may be useful to others, they said this was “because your requirements are very vague, but once you get the research done, then it is useful”.

At a conference two years later, the student introduced themselves saying they recognised the name from e-mails sent during their project. In a short follow-up interview we conducted at the conference the student gave two reasons why they had chosen not to participate. One was that they felt they weren’t able to take on anything extra (see pg 203), the other was that they didn’t see any benefit in it for themselves (see pg 203).

The student said they would have participated if there were marks attached. They were shown some results (Table 18 on pg 195) related to this, showing participants getting improved marks. The student’s response was, “Yeah, fair enough. But my project was much better than my average anyway.” The student reflected on their marks (rounding them up by about five percent in their recollection) and suggested an improvement of 15%. The difference was in fact 11.83%… but still quite high. Asked why there was this difference the student suggested there was a difference in the degree of difficulty of the elective coursework subjects and he had taken hard subjects that were harshly marked. This he felt brought down his average coursework mark below what it should have been. This was examined. The student did do slightly better on their core subjects, but this was only by a couple of percent (2.33%), and their marks were quite consistent. We reject the student’s hypothesis as being inconsistent with the data we have available.

**Observations**

This student’s views of software engineering in research have not changed over the duration of the project. This is unsurprising given they were not a participant. Their reasons for not participating show a clear cost benefit analysis. This case supports the idea of a type of student that is “unable” to participate as they feel they are struggling with the workload already. In this case the students willingness to participate if it was of clear benefit to them somewhat contradicts their reasoning that they couldn’t afford the effort. The case demonstrates that the questions of costs and benefits can not be isolated from each other.
7.5.3 Third Cycle Case Studies
In the third cycle case studies (2005-2006) we introduced the personal process model (pg 162) and put students in control of their research process. A full description of the third cycle can be seen in section 6.2.3 (pg 107).

7.5.3.1 Researcher Case Study 0610
Case 0610 involved a participant and industry partner. The student had a year’s experience working for a start-up company. In the introduction survey (see pg 232) the student indicated they would like to participate because “I wish to organise my project better. Got advantages when report writing”. They felt software engineering did have an advantage for research which included “better planning of the limited time available” and providing “a good basis for the implementation”.

The student had four meetings, but only one FTR, which their supervisor observed. Both the student and the supervisor found the FTR helpful (see page 180). The student also documented their code, but selectively (see pg 171). As the project was with an industry partner, the supervisor said they’d almost certainly be looking at the code during marking.

Observations
This student was interested in SERE but was quite tightly managed by their supervisor. The supervisor in turn did not see software engineering as something relevant to them. While they did not discourage the student’s participation, it is unlikely that they showed much interest and through meeting the student regularly (in some cases as often as three times in a week) the student was kept very busy on the “doing” side of their project leaving little time for reflection. This project showed signs of scheduling problems that were found in the technical review and which the student and supervisor agreed needed to be addressed. In the end this project did run slightly late and a final interview for SERE was not conducted.

7.5.3.2 Research Case Study 0628
Case 0628 was a “reluctant participant”. The student initially stated that software engineering could not help research because there was “too much overhead when you really just need to think and make notes casually”. By the end of the talk they had decided they did want to participate, though gave no reason for this on the form.
Despite their initial impression that software engineering was not fit for research, the student invested significant effort in their participation. The student’s approach was an extreme case of maximizing the benefits to them. At one point in a FTR there was discussion on the benefit of documentation to future researchers, and the student replied “I'm not really concerned with what people do with the code after September the 8\textsuperscript{th}, am I? I mean I couldn't really care less”. The student themselves noted that this sounded selfish but questioned why they should care about benefits others would receive “well why should I, I mean what benefit is there, Ok this sounds selfish…” This student documented according to the guidelines and produced a process plan. The process was inspected in an FTR and it was noted that it contained comments about supervision that were best not shared publicly (the end result of putting the process plan on the web). The student was advised to remove these comments and instead removed almost all the content in their process plan before putting it on the web. This student went on to take up a PhD place in the department.

**Observations**

This student provided an extreme case of cost / benefit analysis. The student was not willing to invest time unless they were convinced there was a chance of it benefiting them, and then they tested it in a small way with a critical analysis of the result. This applied not just to SERE but also to tools ranging from their diagramming software through to their choice of word processor.

**7.5.3.3 Research Case Study 0629**

Case 0629 was a participant who invested a significant amount of effort in SERE. On the introduction survey (see pg 232) the student indicated that software engineering would be of benefit to research as “it is still possible to use software engineering techniques when making the software.” When asked why they wished to participate, they responded that the “tools look useful”. This student has a learning disability that makes ordering information difficult. In addition to the usual benefits, the student found the SERE approach and particularly the process model a valuable coping mechanism (see page 165 for more). The student also found the coding guidelines of particular help (more on pg 174). This student missed the first (code only) technical review but had two other reviews. One of these was observed by a PhD candidate and friend of the student (more on pg 181).
In their post project interview the student was asked what made them volunteer, “To be honest it was a fairly simple decision. It's something that may help me get extra marks and at worst it's going to take up a little bit of time… It's a bit of a no-brainer, which is why I'm shocked that not everyone took you up on it. Do people think they are that much to the bone that they couldn't [participate?]”. The student felt the time they had invested was fairly minimal and was very satisfied with the results (more on pg 202). The student suggested SERE would be most adopted by those wanting an extra edge, but not by the top students, or by the weakest ones (more on pg 191).

This student went on to take up a PhD place in the department.

Observations
This student started with the view that software engineering was the provision of tools, through SERE this view changed to a more process based appreciation of software engineering. The student adopted most aspects of SERE and had a clear motivation for doing so, but still adapted what they used to their own purposes, for example using shorthand in their comments to save time.

7.5.3.4 Researcher Case Study 0625
Case Study 0625 was a non-participant without industry experience. In the introduction survey (see pg 232) they had stated that software engineering would be of benefit to research and that “any piece of software with such high complexity will be likely have a number of complications if SE techniques are not used properly” though they add that “even if SE is used there still are likely to be problems”. The student gives their reason for not wishing to participate saying “I’d rather not put the time investment in and have to spend time learning techniques”.

This case was flagged as interesting based on interview data and again based on an examination of marks. The student was a statistical outlier in terms of marks, and their interview data highlighted the fact that they had implemented many of the practices found in SERE themselves. This included the use of a journal in which they specifically recorded design rationale decisions (more on pg 172), the use of high level diagrams (more on pg 167) and a form of informal technical review (with recommendations) on their design before they started work. The student did not
however have access to the coding guidelines (pg 251) and did not comment their code at all (more on pg 173).

In a survey (pg 241) in the final stages of the project, the student was asked about their research process. The process they describe shows a large element of review by their supervisor. They described their process as “relatively detailed but high level design” followed by review of this design with suggested changes from their supervisor. Next came about eight weeks of coding, and then report writing. The drafting process was done in sections, and the supervisor “would make a number of modifications” to each section. When asked what would have made their task easier, the student replied “Honestly, can't think of anything. Everything went smoothly and the supervision was excellent.”

When asked if they would use more software engineering the student replied “Maybe a little more, just in the realms of developing the components in a different manner”. When asked post project whether software engineering could play a role in research, the student responded “Yes because they can suffer from problems just as any other project can. Surely if using these methods can improve efficiency it would be a good idea.” This is very close to their response in the initial lecture that had taken place about nine months earlier. When asked if they would (with hindsight) have signed up to SERE the student replied “No, it sounds like something that would complicate work. I prefer to just do my own thing even if it might take longer - it's probably just a reluctance to use other people's tools”. This position of a researcher to new approaches is presumably not unique to this student; it is examined in more detail in our discussion on adoption of SERE (pg 188).

After their final interview the student was taken through the SERE website and a discussion about the various process descriptors and tools that were available. It transpired that the student had produced their diagrams in power point, and found both MS Visio and Dia very impressive. At the end of the walk through SERE they commented that “the tools I looked at are helpful, it would be very beneficial to use it a lot of the documents you've just showed me, would have been helpful and useful, documents like the comment one telling you to think about why, what, how, I think
those things would be beneficial to me.” They still found the documentation guidelines intimidating though.

Examining the student’s marks, the difference between their coursework mark and their project mark is between two and three standard deviations from the mean, making this a statistical outlier.

This student went on to take up a PhD place in the department.

**Observations**

This case study is interesting as it shows a number of the ideas implemented in SERE being used independently. This includes the journal, the high-level design idea and a form of technical review (on the design) that appears to have taken place in supervision meetings.

A problem with over-design is highlighted, but this may simply be the student perception. SERE recommends only high level design, and from various surveys and interviews it appears that this is exactly what this student did, and that their supervisor provided feedback that caused alteration to these designs before coding commenced. Without knowing how these meetings were conducted or how detailed these diagrams were, the description is consistent with what would occur in SERE. The student’s impression that a “little more” software engineering might be useful in developing the components seems in some ways to contradict their thoughts about over-design.

This case also provided extensive data on the mind set of those very bright students who chose to go their own way. It is significant that outside of the pressures of the project this student was interested enough in SERE to have a discussion, be introduced to some process descriptors and request access to view the rest at their leisure. This has important implications for adoption, and suggests a risk-free introduction might be taken up by more people.
7.6 Summary

In this chapter we have presented the surveys that gathered data from the students before and after their projects. Various issues previously discussed when examining the literature have reappeared. Issues such as reuse have been discussed at length.

A summary of students preconceived ideas about software engineering and specifically software engineering for the research environment have been presented. We have discussed the tools used by experimental participants and the overall utility of these tools. Some of these tools will be examined in more depth in chapter 8 (pg 160).

Basic approaches such as high level diagrams, journal usage and course code management have been discussed with feedback from both participants and non-participants. Students post project impressions of their own work have also been presented and discussed.

We’ve also examined students’ approach to the research and categorised the difficulties they experiences as essential or accidental difficulties. This reflected back on the idea of improvement through reducing “wasted effort” which occurs on in “accidental difficulties”. The research experience has been discussed in some depth in this chapter.
8 Evaluating Components of SERE

While a lot of data was created and captured as a result of running the experiment, to move beyond an exploratory study it is critical that we change our point of view. In this chapter we focus on some of the more interesting components of SERE and results reflecting on them in their own right. These aspects have been previously introduced in chapter 5, and in this chapter we draw on the data holistically across cases and allow triangulation using data from observations, interviews, surveys, and technical reviews to form a better picture of their impact.

In the data presented in this chapter, the case study numbers have been provided when ever qualitative data from research students has been used. These numbers refer back to section 7.5 (pg 145) where background on the source of the quote has been presented, along with references to other places (in this or other chapters) where data from this student has been presented.

8.1 The Process Model for Supporting Research

The Process Model for Supporting Research was presented as Figure 11 (pg 82). In the course of developing our methodology and carrying out this research three working models of research environment, each an improvement on the last, were created. These were presented in Section 6.2 as Figure 12 (pg 96), Figure 13 (pg 103) and Figure 15 (pg 109). We have also published this work investigating the ideal setup separately (Oboler, Sommerville et al. 2006).

In this section we draw together the key ideas of the Process Model for supporting research. The ideas are presented as a means of understanding and further developing a more effective research environment.

8.1.1 Developments Since the Final Cycle

The idealised model (presented as Figure 11 on pg 82) built upon the third cycle model by incorporating feedback and request for new process descriptors. One process descriptor (the Summary poster, pg 259) was in fact created in this way in the third cycle.
Chapter 8  Evaluating Components of SERE

The ideal model also differs from the third cycle diagram by showing multiple templates. Again in the third cycle a number of templates were effectively used. This was a result of providing those students using the template (and other researchers to whom the template was presented) with versions more in tune with their own research needs.

The concept behind the ideal model is that many researchers would each have their own Personal Process Model (see pg 162), and in it would describe their implementation of the tools and approaches they are using. Different researchers would use different things, and some would in fact not use a process plan at all (as occurs now). The model is sufficient to represent this. The diagram could in fact be made into interactive software to model and catalogue a department’s research development assets (i.e. their process descriptors).

8.1.2 Process Model and CARE

The three stages of model development correspond to increased levels of maturing on our CARE framework (Capability of Academic Research Environment framework – see pg 63). The idea of selecting tools in advance is a move from stage one (initial) to stage two (planned). Not all researchers made this move, non-participants in particular decided not to evaluate the tools on offer. The availability of better ways of assessing the cost and benefit of new tools was a stabilisation at level two of the CARE framework.

The third cycle saw some students begin to development personal research process plans. This caused those undertaking the task to move to stage three of CARE (defined). It should be noted that SERE itself was at this point at stage four, “managed” as it sought to improve the overall research process.

A stage five level of CARE (i.e. optimized) would be possible if research area specific process plans were created and central support was provided (i.e. through the Software Engineering Team) to share benefits and evaluate choices.

Despite the potential for improved maturity, researchers were spread out right across the spectrum. Non- participants were clearly not interested and many expressed the
desire to do things in a familiar way or the inability to cope with the investment of effort required to try anything new. Participants, however, also varied in their adoption. Some never progressed beyond the stay of tool users. They saw SERE as a website where tools could be downloaded and avoided investing in interviews or non software process descriptors. Others used coding guidelines, dOxygen, getting started guides etc, but did not take part in formal reviews or create a process plan.

The stage a researcher was at was evident in the way they spoke about their research and the SERE project. Those with process plans were much more aware of process and much more conscious of the choices they made.

8.1.3 Future work on the Process Model
The process model can be further developed by moving from support for individual researchers to support for research teams and groups. Some aspects of a process plan will be specific to a research area (being similar across universities) others will be university specific (being similar across research teams within the university, but not with similar teams outside). These ideas need to be investigated. Tools to support the development of process plans and the automated recommendation of process descriptors (e.g., people who used this tool also used this set of guidelines) would also be worthwhile areas of further research. These are outside the scope of this project.

8.2 The Personal Process Model
The Personal Process Model was presented in section 5.2.2 (pg 83). Three students implemented the Personal Process Model, all during the third cycle. The small sample is due to it only being made available in the third year and only being introduced to those students who were not only participating, but still actively involved further into the process. This was to prevent the students being overwhelmed with new information and to allow them to get used to dOxygen and technical reviews on their code and documentation before beginning work on the process model.

A sample diagram of the over all research process of one of the participants is presented in Figure 16. This is the diagram after the template has been adapted for the student, and the various sections filled with questions to help them capture critical information and think their process through.
The approach was used not just for the research subjects but for this research as well. A few diagrams for this research project are provided in the appendix. Unfortunately, the idea was only implemented late in our own process and many of the research ideas had already been written up – making their addition to the process plan inefficient. The plan did however prove useful in supervision meetings and greatly eased the task of restructuring parts of this thesis.

Various benefits were realised, in particular: integration with their development process, organisation of information, and communication with others e.g. the software development lab and their supervisor.
Figure 16 The Personal Process Model: An example
8.2.1 The Integration Benefit
Student 0628 explained the benefits of the personal process model as “Clarity, also in terms of neatness and presentation”. He added “If you went back and looked at some of my notes, there's no way I’d understand any of that where as this is understandable to me.” This raises the question of the benefit being purely a result of the transfer of information to electronic form. When asked if a text file would have been as useful, the student firmly rejected this idea, “because a lot of the stuff is related to the code and I tend to write it up during my documentation stage, because it's integrated therefore as part of the eclipse code, yes it is more convenient that way, and more useful. Also as I think about it during my coding, if there are issues to raise with my supervisor if I note it in here I don’t forget about it and as I'm thinking about the code it’s natural to just open that file. In Eclipse I just type [the word] research.” This demonstrates exactly the right use of the process template (as a tool used during coding), with the added advantage of integration into the eclipse environment. Also demonstrated in these quotes is the more holistic approach to managing the research. As the template is no more than a source code file, linking between classes or methods in the template and those in the code is as simple as referring to the class. When processing the input, dOxygen converts the names to hyperlinks.

8.2.2 The Organisation Benefit
In case 0629 the student’s first reaction to the process model was “bloody heck! That's quite a lot of comments”. It was explained that the template was entirely comments except for the little bit of code needed to give it structure. The student summed up what had been done, saying “You've used a piece of source [code] to describe the way you tackle the problem, then exploited dOxygen to generate nice diagrams and things from that information… That's clever, and not immediately obvious.” The student then explained “I have a learning difficulty that effects my memory of sequential reasoning, which means that, if in 10 minutes time you meet me outside and ask to go through everything we've discussed today I couldn't do it, I have all the pieces but can't link them together.” They said the idea of a structured document for storing their ideas would help them. After reviewing the dOxygen output of the sample process plan the student said, “That is a brilliant idea, I love it. I really do.” As the end of the that meeting the student said “the idea of actually documenting the process” was the most useful thing discussed “because [otherwise] the process documentation is usually
done about 2 days before the thesis is handed in and it's [usually] done from memory”.

In their post project interview the student said “I did use the process model, and I probably should have done that sooner. I left it very late and although it was very handy and I got quite a lot out of it for the report, if I’d done that sooner I probably would have got a little bit extra in terms of planning.” The PhD student who observed the technical review on this student’s plans also responded favourably. At the end of their feedback session on the review they were asked if they would consider the Personal Process Model themselves, they replied, “I don't know what I'm going to do (laughs). I mean when I get to a stage that... I would be quite interested in doing this actually. I mean I've heard good things about it.”

### 8.2.3 The Communication Benefit

The final student to use the Personal Process Model was case 0610. Their use was assessed as part of a Formal Technical Review which included their supervisor. In terms of the process plan, it was explained that it “makes the process more systematic” by allowing changes in multiple places to be tracked and monitored. It was suggested that a five minute review of changes to the process plan before a supervision meeting would bring everyone up to speed. The supervisor suggested change logs would be useful to aid this, and these were later implemented by using the Diff program to compare the previously shared version of the process plan and the current version. The supervisor also noted that he and the student had gone through diagrams about twice and regarded this as a breakdown of the process though not a criticism of the model. As was explained, the idea of integration with supervision meetings was a potential idea but was not tackled as part of this experiment – the concept of the Personal Process Model first needed to be tested.

### 8.3 High Level diagrams

In our initial meetings with participants, we questioned whether they had a high level diagram (as was recommended in the “getting started guide”, Appendix pg 247) and in post project interviews we discussed the existence and use of such guides with both participants and non-participants. Data on high level diagrams was also collected through surveys and inspections.

Issues arising from our experiment include: The two potential dangers with design diagrams; over engineering and under engineering. The use of high level diagrams
was shown to be related to reuse, and also arose in technical reviews. High level diagrams are one way of retaining knowledge about the system, specially when combined with auto-generated diagrams based on the code. These issues are discussed in this section.

### 8.3.1 Over Engineering Dangers

A non-participant (Case 0625) highlighted the danger of inefficiency through over-engineering; asked what caused them to waste time, they responded: “too much time spent on design which later on proved itself to be pretty pointless”. This arises when the diagram is too detailed and the details regularly change, as the student explains, “I did a relatively detailed but high level design, this generally consisted of diagrams outlining the structure of components and how they interact. These designs were then shown to my supervisor where he would point out any ways of improvement.” Although this student was not a participant, this is in broad terms the approach SERE would recommend, the difference would be the use of an FTR (pg 176) rather than a supervision meeting and avoiding documentation of details that are subject to rapid and regular change.

### 8.3.2 Under Engineering Faults

In a survey, student 0625 wrote that they had problems with faulty code. This was followed up in a post project interview. Asked if the problem was the result of design fault the student replied “Well it was never really designed, all my design work went in to algorithms, and into components and things, and the design work of actually implementing algorithms was actually done in my head at the time…. It was designed in my head at the time.” This seeming contradiction is clarified by another comment the student made where they described their diagram as “diagrams of the component/class structure, nothing low level”, then later in the same interview as “just an architectural diagram, what components there were, general structure etc”. This suggests that the high level design created was too abstract. The student expressed a similar view suggesting there “were issues with the frame work, which if I'd put a bit more thought into at the beginning I'd have realized this was not the perfect way to do it… the diagrams were perhaps a little too abstract”. This problem did not arise with SERE participants, though it is possible that observations in FTRs prevented a similar situation in at least one case.
A non-participant in the first year (case 0420) wrote in response to a question on whether they had used high level diagrams: “yes... box diagrams!” This shows the gap between what students are taught in class and what they actually apply.

### 8.3.3 Reuse and High Level Diagrams

Data from the first year (Table 6 pg 122) showed that those who used other people’s code typically also used a high level diagram, and felt their work could be extended. Conversely, the data (Table 7 pg 123) showed that those who did not feel their code could be reused, typically did not reuse other people’s code and did not use high level diagrams. This suggests a relationship between high level diagrams and reusability.

### 8.3.4 Diagrams and Technical Reviews

At their initial meeting, case 0610 noted that they had almost got the design done. Asked how they did them, the student replied “Just on a piece of paper. I have various pieces of papers with notes from the reading and one for the design where I doodle UML like diagrams. I haven't used any tools yet, but I'm going to try and do that.” This occurred before they began using SERE and in the same interview the student stated they had not yet read the “getting started guide”. In an FTR for this student later in the process their supervisor was told about the idea of high level diagrams, they commented, “that's interesting because every now and then, it's happened say twice, [the student] has come along with an architecture diagram, some UML, stuff like that. We've had a bit of a sniff with it.” It is unclear whether this was a result of SERE, or the earlier work. What can be said, is that the student was given advice not to go overboard in drawing diagrams with their software tools and this advice was followed. In the FTR the idea of high level diagrams, which are then replaced with more detailed diagrams generated from the code, was discussed. The student’s supervisor interjected with a laugh saying one should then “hope it looks like the original one!” It was explained that this didn’t really matter in research, so long as the result was interesting. The high level diagram is not a contract, just a starting point and a means of focusing effort. The supervisor found this approach acceptable.

### 8.3.5 Knowledge Gap

Returning to case 0625, this non-participant was given a demonstration of the SERE Process descriptors after their final interview. They noted that they had done their diagrams in PowerPoint, describing it as “an absolute ball ache. Every time I do it I
mean to use something else but I never end up doing it.” They were shown Dia and Visio, both suggested on the tools webpage of the SERE project. Asked if they thought this would be faster, the student replied, “much faster, does it do UML?” They were shown UML diagrams after which they said “yeah, well that would make life a lot easier. It would be even easier if one could be automatically generated.” At this point they were shown dOxygen and how diagrams could be generated as a final step to documenting a project.

This case shows that while approaches do exist, students tend to be unaware of them. Although moving far beyond the concept of a website with a “collection of tools”, even simply having tools available and an explanation of what they do would be of benefit to many researchers.

8.4 Documentation and dOxygen

In this section we discuss dOxygen and general feedback about documentation. High Level Diagrams (pg 166) have already been discussed, and the Coding Guidelines will be discussed in the next section (pg 172). It is significant that most participants uploaded dOxygen documentation of their project to the website they built to support their final report. In some cases this was done by students with only minimal involvement in SERE and who otherwise did not take advantage of dOxygen.

8.4.1 Well Documented

The close relationship between the dOxygen tool and documentation was particularly strong in the third year. This was evident in Case 0629 where the student was asked to comment on documentation and replied, “The documentation, well I tie that up with the dOxygen because of course the project documentation was generated using dOxygen through the process model and the documentation for the code was generated using dOxygen, I tie those together.” Not only is the role of dOxygen integrated, but the research and process documentation have also been smoothly integrated. When questioned further, this student said they used dOxygen “to provide good documentation for the project. I already had some comments and I ported some of them to dOxygen format. Enriched it a little bit... the final output was pretty nice.”

8.4.2 Under Documented

In stark contrast we have Case 0420, a non-participant in the first year. When asked what sort of documentation they created for the project the student said they “just
explained each method in the implementation section of the report. Also an explanation on how to run the system on the website.” The only thing the student had heard about SERE was that participants were being asked to comment their code. The student noted that in their own project “the code is not commented”.

Some participants however chose to reduce their involvement as a result of the documentation. Student 0417 replied to an e-mail requesting a date for a technical review by saying “I haven't applied your coding practice yet and I still have a fair amount to write”. As a result this student missed out on the opportunity for a review. This was a first cycle participant and learning from the experience the later cycles emphasised that the review should take place while the coding was in progress and documentation should be done consistently and in small bites, for example half a day on documentation and other organisational work each week (a recommendation that was made to students).

### 8.4.3 Inherited Code

An alternative situation arose when students inherited code. In case 0526 the student started with pre-existing code and commented that “the difficult part might be in the beginning, understanding the code”. According to their estimates they spent a week working eight hours a day coming to grips with the code. When asked if dOxygen could help in this situation, the answer was “Yes because before I have to basically locate them manually. Yes, maybe it would be better.” They added, “there weren't comments all the time… there were some basic comments in parts, but it wasn't commented well.” According to the student, there was a lack of an overview of what the code they had inherited was doing and they suggested “a call graph would be even better” than an overview. The student explained, “I did all these things looking at the code line by line, and jumping there, jumping there, but it took more time.” They felt using dOxygen would have had the potential to cut this initial investment in half. When asked about their own documentation the student replied “My code, in some place there isn't enough documentation. In others it's over documented and confusing. I plan to change this.” In their review, dOxygen documentation generated by the reviewer was presented to the student. The student provided an impromptu walk through of the documentation and was then asked if the dOxygen output was useful, they replied, “yes but I'd also have to look at the code I think, not just something [like
They were shown how dOxygen could include the code and additional diagrams. The response was “Ah nice” followed by a closer look and a comment that “the call graphs were actually very helpful, I didn’t know it [dOxygen] showed you stuff like that.”

A similar situation to Case 0526 occurred the previous year in Case 0405. In this case as well, the student inherited code, only this code had an overview but no detailed documentation (the reverse of the previous case). This student also documented their code to save others the problems they experienced, though reduced the documentation effort as the deadline approached. The student explained, “I used dOxygen to document the code as I was going along, although this tailed off towards the deadline as the pressure to complete increased making me more concerned with just getting everything working.”

8.4.4 Documentation and Technical Reviews

Documentation was also discussed in a Technical Review, with a supervisor and their student (Case 0610). The supervisor described the way they worked as a “standard supervisor student relationship”, saying “once a week or so, we sit at the white board and discuss design and technique”. The question from the supervisor was “do you think those sort of interactions are captured”. The answer provided was that through updating documentation and regenerating dOxygen output for the code and process plan, the world view is always a shared and up-to-date one. It was explained that from a SERE point of view, the initial diagrams gave an idea and then the generated diagrams gave the actual design (though only later on in the process). The supervisor commented that we should then “hope it looks like the original one”. It was explained that from a SERE perspective this is not very important, so long as interesting research is being done. A dramatic change in design is perfectly acceptable and the reason for automatic documentation of program structure is to allow for this without creating an extra burden related to updating manual design diagrams.

In discussing the burden, the student (Case 0610) commented that, “it takes quite a while for me to comment everything” and indicated he “might just stick to the classes where there might be issues”. The supervisor was supportive of this, and the reviewer recommended concentrating in particular areas of the project and to “then start putting
in things you think should be in the report”. The supervisor responded, “personally I wouldn't put in things you'd want in the report, I'd put in information about motivation and architecture. I think that should be in documentation anyway, but that's great, and if we can find a way of reusing that, that's fantastic.” The supervisor when asked if he’d be looking at the code said “almost certainly for this one, given it's going out to industry”.

8.4.5 Further Feedback on Documentation
We asked Student 0629 about the tools in SERE after their project, they replied, “Very, very, handy, specially dOxygen, that's what sold it to me. I can understand that some people would have had need for a few of the other tools, if you had a really complicated project maybe CVS or maybe even a subversion server would come in handy. A few of the other bits and pieces could come in handy for people, but for me it was the dOxygen that had me jumping up and down making life easier.” Case 0629 also commented on the research notebook; looking at the template, the student said “If I’m honest that looks like over kill, but in for a penny, in for a pound”. They did not use the template in the end, opting instead to capture all the details in their process model, the output of which was generated by dOxygen.

Others also commented on dOxygen, journals and other tools. In Case 0507 a student from industry was introduced to dOxygen and responded saying, “It's much better than JavaDoc”, the tool they had previous experience with. In case 0526 the student noted that they had not written their architecture chapter yet and said they would make use of dOxygen in creating this. In the end this student did not use many diagrams from dOxygen directly in the thesis but instead used it to clarify naming, structure and other issues when producing diagrams manually. Case 0625, a non-participant, commented on the idea of a journal. They noted that they had “jotted down notes at the end of every week to take to supervisor meetings but nothing overly detailed/formal”. Further specific comments on the coding standard are discussed in the next section.

8.5 The Coding Guidelines
The guide, introduced in section 5.2.5 (pg 85) was used in all three cycles. In the second and third cycle adoption increased as additional Process Descriptors such as the installation guide for dOxygen (appendix, pg 263) were made available. The
example of a search algorithm and not knowing if it was a first pick or chosen after much deliberation and testing was explained to students (as presented in section 5.2.5).

### 8.5.1 Ambivalence to Commenting

We begin discussing the students’ approaches with Case 0625, a non-participant who in their post project interview said “There's no comments in it. That's what makes it bad. It's well structured, well designed, but with out the comments you're going to be screwed aren't you.” Asked to clarify “no comments” the student replied “none whatsoever, not a line. Maybe a line if you really search.” This was more than hyperbole about poor documentation. When asked if this bothered him, the student responded: “In retrospect yes, because my supervisor and what ever poor sod is going to have to read it as well, they're going to have to struggle with it and seeing as they've just helped me out a great deal it's not very fair not to be helping them out as much as possible”. However, he added “In terms of some anonymous person in Washington or where ever that wants to read it, it's not a massive issue for me, but having said that, just general good practice with a large project you need these reminders just to help yourself and other people. In reality yeah, in retrospect I'd comment everything, yes, it's a good idea”. After looking at the coding guidelines (after the post project interview) the student said “I'd imagine it might be beneficial for some thing like a PhD, but speaking in terms of real research it would obviously help a lot of people, but a lot of it is altruistic help, in an MSc project I couldn't imagine myself going through those rigorous processes. If I did decided I was going to comment… to help my supervisor or whatever I'd comment to a lower degree I'd think.” Later in the interview they added “documents like the comment one telling you to think about why, what, how, I think those things would be beneficial to me. The documentation document is a bit intimidating, but if it was given in bullet points...” Although participants in the first year were given the coding guidelines as a “standard”, it was never suggested that they be followed rigorously and indeed in later years they were introduced as guidelines rather than as a “standard”. The tension between helping yourself and helping others that this non-participant raises was also present in participants who used the guidelines.
A similar ambivalence to commenting, and the tension between selfishness and “good practise”, was discussed by participants 0628. In a technical review looking at the documentation, the student commented, “I’m not really concerned with what people do with the code after September the 8th am I?” In further discussion the student called documentation “an altruistic professional” motivation. In their post project interview however they were far more positive about the benefits to others but still sceptical about the benefit to their own marks saying, “while I claim the benefit in terms of my MSC mark is questionable in terms of reuse there are definitely benefits [including] the way of specifying comments in terms of what this does, why this does and again that process”. At this point the student broke their train of though and commented that “even going back myself and looking at that information has helped me understand what that a class is about and why it’s there and it [(the documentation)] is incomplete and it’s not perfect but it certainly helps.” They also commented that, “It's hard to remember it all [the coding guidelines], which is why I asked for the poster”. The poster they are referring to can be seen in the appendix (pg 259) and provides a one paragraph summary of the coding guidelines and how they connect in with other parts of the process. It was created in response to this student’s feedback.

8.5.2 Adapted Adoption

Another student who used the coding guidelines was Case 0629. The student (when initially looking at the coding guidelines) said “It's nowhere near as bad as I expected. I was expecting some long convoluted process to go through with the coding when really it seems to be very little different to how I like to comment already. Which is good news for me. I should be able to fit in to that with minimal effort.” An observer in the technical review commented that the student “seemed quite enthusiastic about the idea and he always did… To be honest I'm surprised he's not implemented it more.” In a post-project interview the student commented that “the code commenting guidelines have been particularly useful. They really, really, have. They have saved me on numerous occasions from looking at a piece of code and thinking ‘why did I do that?’… The Doxygen was quite helpful, despite the fact that I had to fudge around a bit to make it play with [programming language name].”
Another student (Case 0526) said in their post project interview “Yes, this [documentation guidelines] had an impact, but to be honest I didn’t follow the guidelines, the whole four level comments. Maybe it would have been better, but maybe just following the rules it was better.” They said they took the ideas from the guidelines and used those. The student explains; “Basically you see what the other guy will be interested for, like why, you know all the questions you need to ask, so basically by asking these questions you go and write the comments in the right place.” The “guideline” nature of the coding guidelines was stressed further the following year and students were encouraged to answer the questions raised rather than rigorously follow them.

We examined Case 0526’s compliance with the guidelines in more depth, running through the guidelines and asking the student to rate their compliance. The student gave themselves seven or eight our of ten for clarity and readability, eight or nine in the code for having meaningful names but only two or three in their directory structure and testing. They also rated their comments keeping track of the purpose of a model at seven or eight. Student put comments in classes but not methods, "because I got bored with it, but it's not very difficult. I plan to have comments. It's easier to read a line [of comments] than to read ten lines of code, even if it's not difficult [code]". The student only gave themselves a five for recording rationale as it was “Not in the code, not in all cases”. They rated their internal comments (those not meant for dOxygen or external documentation) at a seven or eight. The student felt that in its current state their code could be rated as four out of ten for the ease some one else could use it. They rated the code they inherited as a five or six out of ten. When asked to confirm they meant their code was in a worse state now the student replied “Yes, because it's more complex and has lots of parts and I haven't documented it properly in some parts.” They said they hope to get it to an eight by the end of the project. The student was asked if they would be using the coding guidelines or another approach, the replied they would use the guidelines but would not be re-reading them as they had already read them. The student said he thought documentation should be done at the end rather than as code was being written and that was his plan for this project. This student did in fact return to the documentation and bring it up to standard, but this is not the ideal way of working.
8.5.3 Impact of the Guidelines
The documentation guidelines were felt by students to have an impact. In the First Cycle they were ranked sixth out of the seven process descriptors at that time (Table 9 pg 126). This is probably due to the presentation of them and the view that they were “requirements” rather than guidelines. In the Second Cycle they were rated the 3rd most useful Process Descriptor out of 16 options (Table 12 pg 132). In the final year they were rated seventh, again out of 16 options (Table 13 pg 136). In the final year while the guidelines dropped, dOxygen which made use of the documentation created through the guidelines, rose to be rated third. This is possibly a result of the introduction of Personal Process Models which also used dOxygen and may have caused students to associate more of the benefit with the tool than with the approach of the guidelines. In both of the later cycle cases, these are very positive results and match the positive responses in other survey questions.

8.6 Technical Reviews
Technical reviews have been introduced in Section 5.2.6 (pg 85). SERE technical reviews began by looking at students code and dOxygen generated output, and after it was created added the process model to the to be reviewed list.

In one review a student’s supervisor participated, in others PhD students observed and then provided feedback on the reviews. In the final year two reviews were carried out on two of the three projects reviewed that year. These included an element of reviewing past reviews (another element of technical reviews). Feedback on the reviews was also collected through observers and the participants.

The interaction mode of our reviews was synchronous though preparation was asynchronous. While the “obvious” approach, this needn’t have been the case. With the availability of collaborative work methods both synchronous and asynchronous approaches could have been used and efficiency rather than physically location was the key factor in this choice.

The Techniques dimension, which can range from free review through to detailed checklists of items, ended up being semi-structured to freeform. In our reviews the Documentation Guidelines provided the items that were checked, though this was
done using the spirit of the guidelines rather than the detail which could easily have been converted to a checklist. In the case of Process plans, again missing details and over descriptive details were the main items being inspected. After this the details of a students approach and factors such as scheduling, samples sizes and other scientific consideration were discussed. The process review was more free-form than the code review.

One approach to examine the perceived impact, i.e. students’ own thoughts on the value of Formal Technical Reviews, is to use students’ post project surveys and the weighted average value students placed on the tools. In the First Cycle FTRs were ranked fourth (Table 9 pg 126). In the Second Cycle they were rated 12th most useful (Table 12 pg 132), and in the final year they were rated sixth (Table 13 pg 136). This variability is largely due to the small number of students who took formal technical reviews. While they may have been very useful to a few students, the weighted average approach looks for the greatest benefit overall. This is indeed shown to be the case if we examine the raw ratings of those students who had formal technical reviews.

The questions from which the ratings are built asked participants to rate various process descriptors on a scale of one to five with the values: one – useless, two – some use, three – useful, four – very useful, five – the most useful bit. Restricting ourselves to the final two years of the experiment (when it was more developed) we must also note that only three of the four users of Technical Reviews returned these final surveys. However, this is a very small sample. Of those who did reply, two rated Formal Technical Reviews as “the most useful bit” and the third rated it “very useful”. This shows a very high degree of value placed on technical reviews. While only three students are used, each of these had a different project and was making a comparison against a large number of other tools. Additional data can be taken from students’ interviews and other comments in their surveys and meetings. In the next sections we look at specific case studies involving or commenting on technical reviews.
8.6.1 Technical Reviews in the Case Studies
The limited adoption, but high value placed on technical reviews makes a further analysis desirable. From the recordings on the reviews as well as post project interviews, we can explore the Technical Reviews and students’ attitudes to them.

8.6.1.1 Timing and Adoption
In Case 0417 in the first cycle a student was given a final reminder about booking an FTR a month before the project was due. The student’s response was “I haven't applied your coding practice yet and I still have a fair amount to write and already have a fair chunk of code so it could take me a while, I will get back to you when I have gone through it all and arrange a meeting!” The student did not follow up, and after the project commented that their project suffered from a lack of documentation and this would make it hard for others to reuse their work. This and similar cases in the first year led to Technical reviews being moved to the start of the coding period and it being emphasised that students were not expected to have finished code for an FTR.

8.6.1.2 Quality Improvements
In the second cycle we collected data on only one review, but an extensive amount of data was collected, which informed our review design for the third and final year. An observer in the cycle two Technical Review (Case 0526) said “It was a good work through. You did quite a lot of going through his problems, and came up with solutions.” The observer highlighted two issues that had been addressed; one relating to a lack of documentation in test cases and the other to a lack of documentation of the modifications the student had made to an open source platform they were working with. The observer said that the review appeared to be “overall very informative” to the student. On hearing that this was likely to be the only FTR in this cycle, the reviewer said “I'm surprised no one... I mean essentially you're giving them a free second opinion on their own work to get their marks up. I'd take that chance.”

During the second cycle review itself the lack of documentation was stressed. The student agreed that with more documentation, “the good thing is that in the end you have pretty good documented code” though added that “for me what I have now is sufficient for me to look back and understand.” The review stressed the importance of documentation in order to share ideas and get feedback from others. The student felt
“it might be considered a luxury” and while great in theory, in practice he felt “it's not going to be that easy to finish a method and also properly document it, and also have time to send it to someone”. The technical review which lasted an hour and a half and ended with the student being asked if he felt the review had been worth while, to which they responded “Of course, yeah, yeah of course. I really like the dOxygen, it seems really nice.” The student added “the ideal thing would be for me to have all of this prepared, all the documentation, but it's quite a task.” On the thought of next year having more reviews more regularly, the student replied “Sounds good, but it's more pressure. It's a small thing, but even I might say ‘oh my, it's another thing’”. While these comments reflect on the cost of having a review, the value was also clearly demonstrated. This showed up when a forgotten to-do note was questioned.

On having a to-do note questioned the student replied, “I forgot to change the information”. It appears both the note and the need for the fix which arose out of changes to a related module had been forgotten. The student commented, “I have to say it's a relief”. A batch of test cases, including regression testing, had been badly named as “test case 1” through to “test case four”. These were given more meaningful names as a result of the review. In the review the student was presented with dOxygen generated output from their code. While they had converted comments to dOxygen format, they had not actually generated dOxygen themselves. The auto generated output was found to be useful not just to the reviewer but to the student themselves (see comments on documentation for further information pg 169).

8.6.1.3 Benefit of an Additional Opinion
After the project student 0526 commented on the FTR in their post project interview, “It's actually good. Up to that moment you basically working alone and you need some technical feedback from someone. It's a good thing that you're not particularly involved in the project, like my supervisor or, what I'm trying to say is that it helps to have a third persons view because a lot of things that you assume someone knows a third person will not know and you need to point them out, and it helps a lot. You know to clarify things, to work on a style so basically you and your supervisor know what's happening, but to make it readable for others as well.” When asked if it was worth the time the student replied “Yes, you know, I didn't prepare all the things you proposed, but I imagine it took more time for you than for me. I think if I had done
more of the things that you had requested, maybe we would have had a better result. It's always helpful to discuss about your project and the issues and get some feedback”.

8.6.1.4 Benefits for Communication
One FTR in the third cycle (Case 0610) was observed by a student’s supervisor. The student commented “The clarification of how I use this with my supervisor answered a lot of questions, basically he asked a lot of questions that I would have asked, your comments were useful. Obviously my process isn't complete at the moment.” He later added that “there's a lot of things we [student and supervisor] need to talk about” that arose out of the review. The supervisor noted that the review was “interesting, it was good to be here”. He added that “to be brutally honest I'm not convinced how useful the technology is, I think the most useful thing they'll get out of it is simply another point of view. You've kept an eye on the documentation as well and gained a separate point of view, that's a lot more useful than pen and paper. You're further way from the project and have a different perspective and that's quite useful.” The discussion ended with positive comments on documentation, though the supervisor felt his area of research wouldn’t “be helped by UML diagrams, it just doesn't fit in that world.” Instead “very good quality documentation, specially at the modular level” was the approach. It was agreed that automatic documentation was better than manual documentation which was out of the question.

8.6.1.5 Benefits for Reuse
In their post project interview, a student (case 0628) said “from the perspective of reusing the project later on the reviews were useful.” When asked if he’s use them for another project the student replied “It would be interesting. It depends on the length of the project, if I wanted it for my PhD... I suppose it makes more sense and has more benefit to make sure things are right and I can understand it. I mean if I come back 18 months, or even a year later and want to understand what that's talking about... for example going through a technical review to find out the problems before I archive it, would be of enormous benefit. Lets say I did a PhD and looked back in my third year on work from the first year, if it's not well documented I would have a lot of problems.”
8.6.1.6 Benefits for Organisation
In a review of case 0629 the observer was asked how they thought the student found the FTR, and replied “He seemed quite enthusiastic about the idea and he always did. When I spoke to him outside the environment shall I say, he did seem very enthusiastic about it. He said it was great, a really good idea and everything. To be honest I'm surprised he's not implemented it [SERE] more.” The student was reminded four months before the end of the project not to leave their review too late. While missing the first review looking at code, they did have a review looking at their process model and how to move to writing-up, it was this review that was observed. In their post project interview the student described the FTR as “very much focussed on how to get from here to the finish line. And how to prioritise things, what to do, what to do when.” While an accurate reflection of their final review in particular, partly due to time pressures of the project, the usual costs and benefits associated with a review still occurred.

8.6.1.7 Reflection and Motivation for Quality
Questioned on the costs of the review the student (Case 0629) explained “It does take a good couple of hours of preparing things to show to you, so that you can get up to speed, but it involves a little bit more time trying to get things straight in your own head, ready to answer questions. Now you can argue that you have to answer those questions sooner or later anyway so the fact that you had to get it straight in your head earlier means you're more practiced at it, but there is quite a time investment for those.” The student also reflected on the nature and benefits of a review “Reviews, they were unusual things, because they were forcing me to partly reflect on what I had been doing and as you say, when I actually sat down with you there were something that I already knew that you had done, because when I had gone through it to prepare for the review I had spotted something and though "ah bugger I'm going to get pulled up on that" (laughs). So there's that element, but then there is also the things that get said that you're not expecting. And those obviously are a greater help because if there is something you knew you really should have done anyway, chances are you would have picked that up come the report or picked that up some where late in the project and at least had an opportunity of doing something about it. The curve balls though, those things that hadn't occurred to me, are even more handy, because they add something different to the project rather than just forcing you to do it properly.” One
such curve ball caused the reviewer and later the supervisor (independently) to recommend the student create an academic paper.

### 8.6.1.8 Scalability

Technical reviews provided a structured way of providing RESET Lab style guidance to a research project. The supervisor who took part in a review of his student’s work commented “I thought it was interesting, it was good to be here,” but added that “to be brutally honest I'm not convinced how useful the technology is, I think the most useful thing they'll get out of it is simply another point of view”. This shows a degree of support for technical reviews including both researchers (including a supervisor) and engineers. Without the support to enable reviews the cost of involving engineers is likely to be too high. The coding guidelines, process plan, and use of dOxygen provided the founding that enabled our re-engineering pattern and allows efficient, yet effective, involvement of limited engineering time.
9 Measuring Success

The aim of this research was to improve the research process carried out by computer science researchers in academia. Having introduced the impact of SERE in Chapter 7, and having separated out some of the components of SERE and discussed them in Chapter 8, we focus now on the overall impact of our approach for SERE.

We begin by defining types of success from the viewpoint of different stakeholders and explain how benefits for the researcher can be used to support benefits for other stakeholders. We then examine the overall success in terms of adoption, formal assessment (the effect on the students grades), the students perception of the costs and benefits, the benefit to knowledge retention and maintainability, the potential to increase the systematic nature of research, the impact on both efficiency and reusability, and finally we examine researchers’ satisfaction with the approach.

9.1 Framework of Success

Improvement can be seen either from the perspective of the researcher or from that of the research itself. There is a point where better research has a substantial cost (usually in terms of time) but gives little reward to the researcher. The underlying principle in our research is to reduce this cost in order to make better research more attainable.

In Table 15 we present the stakeholders in the research process, their goals, how they can achieve them and what factors might mitigate against success. While the table shows a complicated mix of factors, the researcher ultimately has a form of veto power as it is their decision when “near enough” is “good enough” and it is time to move on.
Table 15 Stakeholders and their goals

The “researchers veto” ultimately means that, within the context of our research, efforts to achieve other stakeholders’ goals must not have an overall negative impact on the researcher. Such negative impacts range from “wasting” their time through to decreasing their personal satisfaction. Specifically, process improvement must add greater benefit to the researcher than cost to the researcher; what it achieves on the side for other stakeholders is of little more than academic interest to the researcher.

In our research we have aimed for improvement across all stakeholder groups, keeping in mind the need to make all investment justifiable from a researcher’s perspective. Table 16 shows the approaches that can be used and the benefits to other stakeholders and to the researcher. Many of these are discussed directly later in this chapter.
<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Approach</th>
<th>Benefit to stakeholder</th>
<th>Benefit to researcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
<td>Decrease maintenance burden</td>
<td>Follow on researchers work more efficiently, more timely submissions, Higher publication rate if software adapted to other data sets / purposes</td>
<td>This can save time in software adjustments and help increase publication quantity</td>
</tr>
<tr>
<td>Scientific Community</td>
<td>Increase reusability</td>
<td>Follow on research is easier to do</td>
<td>Software is easier to adapt and maintain, increased collaboration ability</td>
</tr>
<tr>
<td>Scientific Community</td>
<td>More systematic approach</td>
<td>Replication experiments become easier, quality improvement</td>
<td>Increases productivity, allows planning to mitigate problems</td>
</tr>
<tr>
<td>Scientific Community</td>
<td>Increase knowledge capture</td>
<td>Increased contribution to the body of knowledge, saves other researchers time</td>
<td>Crystallisation of ideas to aid creation of publications</td>
</tr>
<tr>
<td>The Public</td>
<td>Reduced PhD completion times</td>
<td>Increased through put, more efficient use of resources</td>
<td>Faster completion for PhD students, increased through put = more major publications for supervisors</td>
</tr>
<tr>
<td>The Public</td>
<td>Facilitate reuse</td>
<td>Lower cost research</td>
<td>(see “increase reusability”)</td>
</tr>
</tbody>
</table>

*Table 16 Researcher’s and other stakeholder’s benefits*

In addition to the approach in Table 16, both the institution and the researcher want to increase the number and quality of publications. This is facilitated through the approaches: “Increase knowledge capture”, “Increase reusability” and “more systematic approach”. In these and other results we are able through our experiment to measure the benefit to the researcher (at least from their own point of view), but not to the other stakeholders.

Our inability to measure the benefit to non researcher stakeholders is due partly to our use of MSc students and short projects as test subjects, rather than for example PhDs or researchers on grant funded projects, and partly to the time constraints of this project. Real benefit to the public, institution or the research community would require implementation of our research across a department, coupled with a longitudinal study. We hope this research sets a foundation for such future work.
In the remainder of this section we examine the benefit to our research participants both as students wanting a higher grade, and as researchers looking for some of the benefits outlined above. We begin with a discussion on adoption, as improvements researchers will not accept have little benefit. Next we discuss the overall benefit to our participants in the most concrete and independent way possible – as an effect on their own grades. In the remaining sections we discuss benefits we expect our approach to provide and examine the evidence for or against these benefits based on participant and non-participant responses to surveys and interviews as well as our own observations of students work.

9.2 Adoption of Sere
The most basic measure of success is adoption. People will choose those conditions they expect to provide them with the greatest reward for the lowest cost. Under pressure this tendency increases as subjects become more risk-averse.

In our experimental setup there was a clear separation between the evaluation of the experiment and the grading of academic work. This was made clear in explanations, and through the separation of the experimental support role and the supervisor role (carried out by different people and at different times). This removed the potential bias of markers being influenced by students’ participation or non-participation, or of students participating in the hope of gaining favour with markers.

Before the start of the first year’s experiment, students were asked why they wished to participate or not participate. Responses for participating focused on the perceived benefit of learning how to “structure” or “engineer” their project work, the chance to learn “skills” and “software engineering techniques” and the opportunity to receive “guidance” or “help”. Two respondents said that they felt they should participate due to external pressure (supervisor / e-mails sent to them). Conversely, those who chose not to participate cited “time” and a concern over the commitment involved. These results were reinforced by post project interviews with both participants and non-participants. Researchers do perform a cost / benefit analysis when considering new approaches or tools and the cost is largely dependent on time, though we also discover a limiting element related to the amount of new information, some students feeling they simply couldn’t cope with any additional ideas.
In Table 17 we show the division between those who chose to participate and those who chose not to. Our definition of participation is based on having an account to access our password protected website (which provided our tools and resources). As shown participation was about evenly split each year. Participation at this level does not distinguish between active and more passive involvement.

<table>
<thead>
<tr>
<th></th>
<th>Participant</th>
<th>Non-participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year One</td>
<td>7 (experimental)</td>
<td>8 (control)</td>
</tr>
<tr>
<td>Year Two</td>
<td>12</td>
<td>12 (+1 outlier)</td>
</tr>
<tr>
<td>Year Three</td>
<td>11</td>
<td>13 (+1 outlier)</td>
</tr>
</tbody>
</table>

Table 17 Population of samples over three years

In post project interviews we further examined students’ choices on participation. We asked why students chose to participate (or not) and for the characteristics of someone who would make the same choice as them.

9.2.1 A Non-Participant’s View: Case 0625

We asked Student 0625 why they chose not to participate, how they felt in hindsight about participation, and what would have convinced them to take part. After the final interview we introduced them to the SERE website and the process descriptors on it and sought their views. This turned out to be a very interesting case both in terms of the interview and the student’s marks. The interview took place after his final presentation but before marks were released. The marks showed that this student did significantly better on his project than on his coursework (the improvement being between two and three standard deviations from the mean).

Asked why he didn’t participate, and looking back whether he would participate if starting over with hindsight, the student replied, “It all went smoothly and I wouldn't want to change that… I thought it might be a little bit rigid for me. I'd rather just do what I want to do and get it sorted the way I want to rather than try another method, which is how I've always done it. I'm probably just set in my ways really. It's the old person type of thing, I don't want to learn how to use a computer, I don't want to know how to use a central heating system.”
The interview highlighted that he had unknowingly done some of the things participants were asked to do including a high level diagram and a journal. The things he did not do related to documentation where he stated that the design was done “in my head at the time” and on source code documentation he said “there's no comments in it. That's what makes it bad. It's well structured, well designed, but with out the comments you're going to be screwed aren't you.”

The student was asked what would convince them to participate. They said they’d be willing to listen if it was presented in the context of a lecture, but only to listen, not to sign up. They then said, “I don't know you’re talking to someone who is very set in their ways. I want to do it my way. If you'd said there's no mandate on doing things or how you do things, halfway through if you decide you don't want to do it, pull out... I really don't know what I'd say to be honest. I’m just one of those types of people that this is how I'm going to do it, I can't be bothered to learn a new way of doing something even if it is actually better. Does that make sense?” In actual fact once they signed up, students were able to opt in or out of any part of SERE. Given the student would soon be starting a PhD they were asked if they’d consider looking at SERE for their PhD, they replied that with out the time pressures they’d definitely be more willing to look at it. On the other hand with a time limit and particularly on something they were finding challenging they would be unlikely to try SERE, e.g. “If it was hard and I knew I was going to struggle with this one, I can't imagine me taking it up”.

When asked for ideal conditions for SERE participation the student replied it would benefit “people who are not too set in their ways but are experts in that area. [With a project that is] lengthy and not over challenging so people don't think it's too short so they won't benefit, or it's too complex and we have so much to learn already that it's not worth trying all this.”

Asked to give a profile of a student like themselves, one who wouldn’t really look at SERE and just wanted to get the project done their way, the student replied “without putting myself down, lazy, and generally don't want to try new stuff because they think it's going to be more work- which is one of the properties I attribute to not doing it. Possibly a smart asses, I know better than this guy I'm not going to try something new, not considering myself as one of those. People who are out of the field of software engineering. I don't really enjoy software engineering courses, when I
approach things I'm more of networking kind of person. To be honest this project has changed me. The use of components etc have set me as a very well ordered structured person. Previously to that you'd be looking at a stream of crap (laughs) in a load of class files. That could be an advantage or disadvantage, some people like myself wanted to clean my act up, so you can appeal to those sort of people on that sort of aspect.”

After reviewing SERE’s process descriptors and discussing whether it should be made available to the following year’s students, this student strongly supported making SERE available, despite not using it themselves. They said, “I'd have modified my project or at least how I'd done certain aspects of it e.g. paid more attention to commenting, it would definitely benefit people.” The student added, “I would like to browse through and pick out bits I want, as I said before the main reason I didn't was laziness. That's the main thing for students, seeing it as extra work on top of stuff they already have to do, which they already see as a substantial amount of work.” This shows a gap between resistance to formal participation and commitment and a willingness to adopt if it was introduced and then left to them to pick what they liked at their own pace and in their own time. The student commented that being introduced to SERE process descriptors in class would be “ten times better” than simply giving people access to the site.

9.2.2 An Inactive Participant’s View: Case 0529
Another type of non-adoption comes from participants who sign up, but then fail to use SERE process descriptors. The student in Case 0529 was a participant who initially invested the time to learn about the tools and attended a first meeting. They later explained “unfortunately I couldn't really use it, it was very interesting, from the first meeting we had, it would have helped me a lot specially with my documentation, specially with tools for diagrams... but unfortunately with errors, changes and problems cropping up in the project [I wasn’t able to].” The student explained their reference to problems saying “from the beginning the problem was with the scope” and having not “judged quite accurately the difficulty and whether it was doable or not.” They said they underestimated the degree of difficulty and mentioned they initially lost about a month (which they commented was “quite a lot”) going down a wrong path.
The student said, “I finished on the last day. There was a lot I didn’t have time to do, also in the writing up. Because there were many changes at the end, there were many things I didn't have time to put in my report that I wanted to.” On SERE they noted “I knew some things that were available, certain guidelines I would have to follow... I knew some of the things that would be helpful.” Asked if they would use it given hindsight the student said “If things were a bit more under control on my part then I think I would have used them.” When asked whether they would have participated more (with hindsight) if the problems were the same the student replied, “It might have been helpful, I don't know it's not the path I took.” They added that perhaps some things would still have been useful with hindsight.

This student demonstrates a case where all the steps needed for adoption did in fact occur and the student was convinced of the benefit and planning to adopt, before outside influences and time-pressure forced them from the adoption path.

9.2.3 View of Participants
In this section we present five cases of participants and their comments on adoption.

Student 0628 was asked in a post project interview which aspects of SERE they might use in their PhD. Their immediate response was “dOxygen might be helpful”, when asked about an FTR they replied “No but that's just me, another person might, that's just me”. Asked about their willingness to participate in someone else’s FTR, the student replied “Depending on the time of year and the amount of work I had, sure, it would be interesting, it would be fun”. Further into the interview they added that they “might use some of the process documents, yes... I might use them in the future. DOxygen maybe. Coding standards, probably.” When asked what they meant by “probably” the student replied “it just depends if I'm lazy”. Asked if they’d recommend participation to future students if the experiment ran for a hypothetical 4th year the student replied, “I'd say yes, but I would give them some advice, I'd tell them to not take it too far but to customize it to suit their own needs. To look at the tools but also consider other tools. To keep a poster and try and understand what is going on. To ask you for clarification on things they don't understand. To integrate your process document into your IDE if possible.” This student showed a high level of adoption
despite noting they might in the future be lazy. There are similarities in attitude with Case 0529, although this student was more willing to consider new approaches and if they worked, to adopt them.

We asked student 0629 to describe the characteristics that would define a group of students like themselves, and the response was: “those who would try anything to perform just that little bit better, who are always looking for that extra edge. But that is because of the way it was presented to us. It was presented as an experimental thing that might help you. And the type of people who go for that are the type who are just pushing to try and see if they can get that little bit extra.” Asked what sort of people were likely to participate in, and adopt SERE, the student replied, “In my experience there are always a couple of people in every class who are very, very clever and know it. They don't tend to apply themselves too heavily to things, they breeze through knowing that they'll be just fine. I think those people are less likely to do the SERE, I think it's more the people who have got to work. Why would they [the top people] want to spend extra time doing something they know they're going to do just fine with anyway? Of course they could always be wrong. But the people who have to, as I put it, scratch and claw their way above the herd to try and get towards the top, the people who don't have it easy is probably the best way to put it. It's going to be the people who don’t have it easy who try the hardest to get every last mark.” The view of those who might not bother participating resonates very strongly with the views of student 0625 who was just such a participant.

Student 0629, when asked their thoughts on the usefulness of SERE for PhD students and academics, replied “academics time is very limited, they may have additional hang-ups about investing extra time to do things that really they don't want to do. And the PhD students, I don't know there is kind of an expectation that they would be doing most of this already I assume.” This faith that others are using best-practice does not stand up to scrutiny, as we have shown in previous work (Oboler 2003), and demonstrated with respect to technical reviews in Table 4 (pg 86). Further evidence can be found in the general lack of papers examining the academic research process in computer science. The student was asked what they might use in their PhD (if anything) for SERE, and they replied “Documentation guidelines definitely, probably the process guidelines as well. I'm afraid not the endnote or anything like that. I'm still
happy with my haphazard research filing system.” When asked about reviews the student said in a PhD situation they wanted to have one “every 6 to 8 weeks probably. That would be the happy medium, there would be a lot of progress in 2 months, but regularly enough for you to be pulled back if you screwed something up or there was a gaping hole that needs plugging.”

Another aspect of adoption is promotion by experts. Student 0506 said that “I don't think [SERE] was pushed enough... I don't think I heard many lecturers actually mention it.” The student goes on to say, “there are tools there that I think are extremely good for people to use, but they've got to get to know them. I know people assume that if you're reached this level you can go around and find things, never the less a few adverts would be beneficial.” This student’s comments suggest they feel the approach should be standard, but they also suggest that there was not undue influence encouraging students to participate. Coming from a part-time student this view is a reflection on both the second and third cycle.

Student 0507 explained their desire to participate in their initial interview by saying “I think I can program after many years experience, now I want to progress beyond. That's why I'm so happy to meet you and to work on this project”. This shows a desire not just to “get through” the Masters degree, but also to learn as much as possible. This student had been sponsored by their employer to return to university and had a secure job waiting upon their completion. The project itself was related to their industry work and subject to commercial confidentiality though they and their sponsor were happy about inclusion in the SERE experiment.

Student 0526 tried discussing SERE with friends on the course, “[Another student] said it was helpful, some other guys might have the response, ‘I'm up to my head with the report, I can’t deal with extra work’, this was a classic response.” The student saw participation as positive and cited many benefits both holistically and for parts of the work. Asked if SERE was helpful they replied “Yes, always having another point of view is very helpful. [It] also helped for organizing the work.”
9.2.4 Further Discussion
Case 0628 has much in common with case 0625, both describe themselves as lazy, and both are at the very top end of the class marks-wise, one topping the class overall and the other receiving the top project mark. Both match the description of those unlikely to participate, as given by the student in case 0629. Both are cynical about new approaches and tools and require significant proof before investing time. The difference between them, that led one to sign up and the other not to, is that case 0628 had an interest in software engineering and enjoyed exploring new tools. Case 0625 in contrast preferred to stick to what they knew, even if they knew this was a less than ideal approach. This is exemplified by case 0625’s use of PowerPoint to create diagrams compared to case 0628’s use of Open Office and their search of bibliographic software that would work well with it. The implication is that high levels of adoption would be best achieved through incorporation into the learning environment before students begin serious research. Senior researchers may either be explorative and open to new approaches or fixed in their ways, preferring to be knowingly inefficient but consistent. Members of the software engineering community are likely to be early adopters due to the characteristics that cause people to take an interest in software engineering.

9.3 Formal Assessment (Effect On Grades)
The choice of MSc students for our experiment allowed us to compute a utility heuristic based on expert opinion, that is the grade assigned to them by academic staff. We compared this heuristic between the participants’ sample group and the non-participants’ group.

While we do not have enough data to reach a scientific level of statistical significance (i.e. 0.05) the results of the heuristic comparison provide supporting evidence that is inline with other observations. The data presented here is an amalgamated collection from the second and third year of the experiment. The results may be combined as the variation between participants’ degree of involvement (within each year) was seen to be far greater than the variation resulting from changes to the experiment between these cycles. At the end we discuss the first year results as these resulted from a markedly different method to the other years and significantly impacted on our experimental design.
9.3.1 The Improvement Heuristic

Our experimental methods are only one of many factors that may affect a student’s mark. Without accounting for each student’s natural level of ability, a comparison between students (or indeed groups of students) becomes meaningless. Instead of relying on the raw marks students received for their project, we have instead deducted the student’s average “coursework” mark from this. The coursework mark is the total of all their other marks (core modules and electives). The resulting number may be either positive (the student did better on their project than in their coursework) or negative (the student’s project performance was below their average coursework mark). This can be expressed as:

\[
\Delta = p - \left( \frac{1}{n} \sum_{i=1}^{n} c_i \right)
\]

Where \( p \) represents the project mark, \( n \) represents the number of subjects a student did, and \( c_i \) represents the coursework mark this student got for subject \( i \).

For simplicity we will refer to this change-heuristic as \( \Delta \) (delta) elsewhere in this thesis. The \( \Delta \) metric removes many of the variables that are consistent for students through the year. This includes not only their natural ability, but also language difficulties and environmental factors. Significant variables that are not taken account of include: student-supervisor relationships, illness or other difficulties during the latter part of the year only, research problems, and the level of participation in our experiment by the student. Some of these factors may cancel others out, and some may exaggerate other factors.

We do not expect to see a consistently positive \( \Delta \). To show that our methods are an improvement we need only show that the average \( \Delta \) has improved, and preferably that the distribution is both more positive and more consistent.

As our sample is self-selecting, there is a chance of sample bias, i.e. that student who would naturally do better in the project signup. The use of the \( \Delta \) metric limits the effect of this form of bias by measuring improvement not raw ability. In our comparison of the groups we examine the risk of such bias in more detail. We do not believe sample bias played a significant role in our results, except that students who
were struggling were unlikely to sign up for anything additional; this point has already been discussed in the adoption section.

9.3.2 Preparing the Data
The data presented here is in four parts. In the first (our largest data set) we combine the results from the second and third year of our experiment. With the larger sample we also include two outlier results, both for non-participants.

These outliers (one at either end of the spectrum and of about the same magnitude) sit between two and three standard deviations from the norm. In the appendix (pg 312) we include a copy of the combined data, less the outliers, for comparison. The difference is very small.

In the second and third parts we present each of the final two years of data on its own and with the outliers excluded. As the outliers occurred in different years, they would skew the data sets (for non-participants) in different directions making comparison difficult. The case of the outlier that is a significant improvement on the average, is partly explained through additional data we collected on this case.

In the final part we examine the first year’s data which was collected under a different experimental setup and before our methods were well developed. This data was not included in the combined results as it represents a significantly different situation.

9.3.3 The Combined Data Set
In this, our largest data set, we introduce the summary to the data in Table 18, and then provide a detailed comparison of the participant and non-participant group’s performance using the heuristic metric $\Delta$.

<table>
<thead>
<tr>
<th></th>
<th>Mean Proj</th>
<th>STDEV Proj</th>
<th>Mean CW</th>
<th>STDEV CW</th>
<th>$\Delta$</th>
<th>STDEV ($\Delta$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>63.98</td>
<td>9.50</td>
<td>60.88</td>
<td>6.45</td>
<td>3.10</td>
<td>7.32</td>
</tr>
<tr>
<td>P's</td>
<td>65.23</td>
<td>8.58</td>
<td>60.19</td>
<td>6.89</td>
<td>5.04</td>
<td>5.60</td>
</tr>
<tr>
<td>N's</td>
<td>62.91</td>
<td>10.26</td>
<td>61.47</td>
<td>6.13</td>
<td>1.44</td>
<td>8.26</td>
</tr>
<tr>
<td>Change (P-N)</td>
<td>2.32</td>
<td>-1.67</td>
<td>-1.28</td>
<td>0.76</td>
<td>3.60</td>
<td>-2.66</td>
</tr>
</tbody>
</table>

Table 18 Summary data for 2004-2006
An overview of our sample is shown in Table 17. The comparative table with the outliers excluded can be seen in Table 39 (pg 312). The table shows that on average
participants had performed slightly worse in their coursework, but slightly better in their projects. The $\Delta$ value of 5.04 for participants (meaning about a 5% grade improvement) is both valuable from the students’ perspective, and noticeably larger than for the non-participants.

The median for $\Delta$ values are All: 3.30, Participants: 4.33 (interquartile range: 9.64), Non-participants 1.42 (interquartile range: 11.00). These results are similar to the means. We can further examine the medians using a box plot comparison of the two samples. This is shown in Figure 17.

Figure 17 shows that participants’ $\Delta$ were spread over a smaller range and tended to have a more positive $\Delta$. The lower quartile of the participants sits comfortably around the mean of the non-participants’ data. The mean of the participants’ data rests high within the upper quartile of the non-participants. As 50% of the data rests within the boxes in a box plot, we can see that participants do better on their projects compared to their coursework not only on average, but also far more frequently.

The difference between the two populations can also be examined on the variation between their means. This is done using a T-test in Table 19. The tables shows that at the 90% confidence interval, the mean of the participants and non-participants group are significantly different (i.e. a variance of 0 is not within this range). At the 95% confidence interval this is no longer true, the range there is from $-7.69$ through to $0.49$, so a variance of 0 would be unlikely, though not statistically significant.
Table 19 Comparison of means (T-test)

The results strongly favour the hypothesis that the treatment (participation) had a positive effect on participants, but they are not strong enough to confidently reject the null hypothesis. This is expected due to the influence of other factors and the small sample size in statistical analysis terms. The additional data collected through case study methods (surveys, observations, interviews etc) add to this data allowing us to form a richer picture and gain confidence in our results. While small for statistical analysis the size of the data set is large for an experiment with in-depth analysis.
Given the time constraints of this research and the investment per student the sample size is thought to be at an optimal level and these results are very encouraging.

9.3.4 The Third Year's Data
In the third year (Table 20) we can see non-participants doing better in their coursework than participants, yet participants doing better in their projects than non-participants (reversing the gap and then edging ahead).

<table>
<thead>
<tr>
<th></th>
<th>Mean Proj</th>
<th>STDEV Proj</th>
<th>Mean CW</th>
<th>STDEV CW</th>
<th>Δ</th>
<th>STDEV Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>65.58</td>
<td>8.51</td>
<td>61.88</td>
<td>6.34</td>
<td>3.70</td>
<td>6.67</td>
</tr>
<tr>
<td>P's</td>
<td>66.91</td>
<td>8.84</td>
<td>60.97</td>
<td>7.58</td>
<td>5.94</td>
<td>5.97</td>
</tr>
<tr>
<td>N's</td>
<td>64.46</td>
<td>8.41</td>
<td>62.65</td>
<td>5.27</td>
<td>1.81</td>
<td>6.86</td>
</tr>
<tr>
<td>P-N delta</td>
<td>2.45</td>
<td>0.42</td>
<td>-1.68</td>
<td>2.31</td>
<td>4.13</td>
<td>-0.89</td>
</tr>
</tbody>
</table>

Table 20 third year delta improvement data
The Δ scores in Table 20 show an improvement in both groups, but a much more pronounced one (almost a six percent improvement) by the participants. Participants had a wider standard deviation in their coursework marks (7.58) compared with non-participants (5.27). In project marks, however, the standard deviation between the groups is fairly similar, and higher than for coursework. Of note, the standard deviation of the participants is lower than the non-participants, despite a much higher Δ. This suggests not only a marked increase in delta for the treatment group, but also a more uniform change than for non-participants.

9.3.5 The Second Year's Data

<table>
<thead>
<tr>
<th></th>
<th>Mean Proj</th>
<th>STDEV Proj</th>
<th>Mean CW</th>
<th>STDEV CW</th>
<th>Δ</th>
<th>STDEV Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>62.75</td>
<td>7.84</td>
<td>60.14</td>
<td>6.59</td>
<td>2.61</td>
<td>6.10</td>
</tr>
<tr>
<td>P's</td>
<td>63.69</td>
<td>8.43</td>
<td>59.47</td>
<td>6.45</td>
<td>4.23</td>
<td>5.35</td>
</tr>
<tr>
<td>N's</td>
<td>61.81</td>
<td>7.45</td>
<td>60.81</td>
<td>6.94</td>
<td>-0.53</td>
<td>7.50</td>
</tr>
<tr>
<td>P-N delta</td>
<td>1.88</td>
<td>0.98</td>
<td>-1.35</td>
<td>-0.50</td>
<td>4.75</td>
<td>-2.15</td>
</tr>
</tbody>
</table>

Table 21 second year delta improvement data

9.3.6 The First Year's Data

<table>
<thead>
<tr>
<th></th>
<th>Mean Proj</th>
<th>STDEV Proj</th>
<th>Mean CW</th>
<th>STDEV CW</th>
<th>Δ</th>
<th>STDEV Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>61.47</td>
<td>6.73</td>
<td>62.35</td>
<td>5.61</td>
<td>-0.88</td>
<td>5.33</td>
</tr>
<tr>
<td>P's</td>
<td>62.55</td>
<td>6.52</td>
<td>63.07</td>
<td>6.03</td>
<td>-0.52</td>
<td>5.44</td>
</tr>
<tr>
<td>N's</td>
<td>60.83</td>
<td>6.96</td>
<td>60.94</td>
<td>5.74</td>
<td>-0.10</td>
<td>6.78</td>
</tr>
<tr>
<td>P-N delta</td>
<td>1.72</td>
<td>-0.45</td>
<td>2.14</td>
<td>0.29</td>
<td>-0.42</td>
<td>-1.34</td>
</tr>
<tr>
<td>PE's</td>
<td>62.39</td>
<td>6.41</td>
<td>61.06</td>
<td>7.79</td>
<td>1.33</td>
<td>3.75</td>
</tr>
<tr>
<td>PN's</td>
<td>62.75</td>
<td>6.61</td>
<td>64.74</td>
<td>2.88</td>
<td>-1.99</td>
<td>6.10</td>
</tr>
<tr>
<td>PE-PN delta</td>
<td>-0.36</td>
<td>-0.20</td>
<td>-3.68</td>
<td>4.91</td>
<td>3.32</td>
<td>-2.35</td>
</tr>
</tbody>
</table>
Table 22 First year delta improvement data

Table 22 shows the 2003-2004 results in a compatible format to the latter years. As discussed previously (pg 96) the methodology for the first year involved a control group (equivalent to other years) (N in the table) and an experimental group that was itself split into two, one receiving full support (Participants Experimental – PE’s) and the other only having website access and receiving reminders via e-mail (Participants Non Experimental – PN’s).

This data sample is unusual as the randomly assigned PN group has a higher coursework score than any other group. This was not known until the projects were completed. The sample sizes within the first year are also small; 14 non-participants, seven experimental and seven non-experimental. The box plot in Figure 18 shows the spread of the results within each sample. As can be seen, some strong negatives pulled down the PN scores. The PE’s performed much like the experimental group in other years.

The non-experimental group did worse on their projects than on their coursework and ended up with around the same mean project mark as the experimental group. Although this could be entirely due to a few bad cases, it is possible that participation would have prevented these problems. After receiving these results and students’ feedback, it was considered unethical to ask for volunteers and then leave some of them exposed to an unnecessary “risk”. This resulted in the change to methodology eliminating the non-experimental participant group. Neither the experimental group in the first year, not in the subsequent two years showed data with similar problems. Either the first year was an anomaly or the treatment, or possibly larger sample sizes (relegating such drops to outliers) eliminated the problem.
9.4 Students’ Perceived Costs and Benefits

A distinction must be drawn between the actual benefit to student, for example an average improvement in their marks, and the perceived benefits such as the time the student felt they had saved, or a perceived reduction in the complexity involved in managing their project. These perceived benefits are based purely on qualitative data and taken from the student’s point of view.

9.4.1 Case 0405
One of the students in the first cycle (Case 0405) expressed concern that while benefits might exist, they would not be the one collecting them, then explained; “the benefit is hard to measure, and people will naturally want to measure the benefit that they will receive for their effort. I’m not sure however that the major benefits reaped for the effort will be for the person who puts the effort in. Software engineering will provide clear benefits for people looking at the code, and will probably help projects that require considerable design more than those that don't. I’m not sure how much benefit I personally will see from commenting my code, unless the supervisors are
bothered to read any of it (unlikely). The clear benefit of this is for future users of the code, and not myself (a selfish outlook, but that's just the way it is). Having said this, structuring the code as I did benefited my understanding of what was going on as I worked, although as I said previously I naturally structured the code as I did without the additional help (largely at least)." The main benefits this student said they received from participation were the use of dOxygen output, and they commented that they were “pleased to be able to make use of dOxygen to produce formatted code segments to include in the write-up though - wouldn't have known about it otherwise”, and although they said they would “naturally tend towards structuring the code” as they did, the “code comments would have been more limited and I would not have attempted to use dOxygen or related software to produce code documentation.” While the benefit was there, the cost associated with the coding guidelines and dOxygen was much higher in this first year and as such the benefit (compared to the cost) less clear.

9.4.2 Case 0628

In their post project interview, when student 0628 was asked if he’d participate again (with hindsight), he replied “I'd say yes, but with alterations. I did do it with alterations, but sometimes you feel guilty or I include something to make you happy even if I'm not sure about the benefit.” When asked if any of these things paid off the student replied “Yes, well I'd have to answer maybe. I'm pretty sure, but it's hard to remember which bits I was unsure about.” Reflecting on the process as a whole they said “having dOxygen makes a big difference, having some sort of overview of your design and description of what you’re doing makes a big difference.”

The student (case 0628) was also asked to enumerate what was useful in SERE and what had been a waste of time. They replied, “Useful, clarifying the comments and problems I had in there, talking me through changes. Explaining parts of the process documents. It gives you a level of reassurance.” The student noted that having a supervisor look at the code would be very useful but “would bore the supervisor silly”. In response to the waste of time question, the student replied “depending on your point of view whether you're talking about getting the marks at the end of the MSC or in terms of reusing something. In terms of reusing something, it's not a waste of time. It is mostly if not all useful. In terms of getting marks on my report, I put a big question mark on that, I don't expect them to actually spend time looking at
documentation etc.” In terms of impact they sated “It [the process model] definitely had an impact in terms of recording my meetings and miscellaneous information… but it was more useful outside the technical to explain the more general process, that was helpful.”

Rather than a Hawthorn effect (see pg 40) this student seems to be going out of their way to be contrary (this could be said to be part of the nature of this particular student), yet they still provided positive feedback. In defining types of researcher this student and those like them could be termed “active sceptics” for their ability to engage, but more to prove something is inferior to their existing practice than to adopt a new approach. Where they see the benefits they are however converted.

9.4.3 Case 0629
In Case 0629 the student summed up by saying “it's useful, it's helpful, it does involve extra time, but it does pay you back. It does pay back, it does save you time, yes. It's well worth doing.” They also commented that “the code commenting guidelines have been particularly useful. They really, really, have. They have saved me on numerous occasions from looking at a piece of code and thinking ‘why did I do that?’… The dOxygen was quite helpful, despite the fact that I had to fudge around a bit to make it play with [programming language name].”

When asked why they chose to participate they replied “To be honest it was a fairly simple decision. It's something that may help me get extra marks and at worst it's going to take up a little bit of time. The results from previous years looked promising, you know people who'd done SERE tended to do a little bit better, so invest a bit of time and get a few marks, or don't invest and plod along and don't get the extra marks. It's a bit of a no-brainer, which is why I'm shocked that not everyone took you up on it. Do people think they are that much to the bone that they couldn't, I mean what, the entire contact time between us for this has been what, 5 hours? Plus the time I spent with the tools... that's essentially one day of work… And of course of that is spent at an earlier stage. It wasn’t a day right at the end when I was to the bone, it was a day right at the beginning, or at least part of it was anyway.” It needs to be stressed that while the recommendations to incoming students (from past participants) were passed on each year, information relating to marks was not passed on and much of the mark
analysis was only completed after the final year’s marks were received. As noted by case 0628 (directly above), there was a level of doubt on the influence SERE could have on students’ grades.

The student (Case 0629) was asked what he’d say to student who think they can get by perfectly fine on their own, don’t need any help, and don’t want to put in the extra effort. In short a student matching the profile of case 0625. The student replied “Well I could have got through the project perfectly well without SERE… the work would probably have been 90% the same. I'd have to rush the end of the report in order to do a website, and I would have perhaps had a bit of delay during coding when I had to look back at things because I hadn't followed the documentation guidelines, I wouldn’t have had the cool process model to show people, but I would have got through the project ok. It's not that you need this SERE thing to do it, it's that if you’re willing to invest the time, you can get yourself an extra couple of marks, and a better project. I mean I would have been able to coast this project and get a decent mark, say 60 odd something.” When asked why they didn’t do this they explained “my average mark for the year is 68 point something, if I creep in and get a top mark for this project, I'll sneak a distinction overall. But it's a tall order… I don't think that's actually going to happen, but might as well have a bash.” This student did indeed manage enough improvement in their project mark to bring their average up to a distinction level.

9.4.4 Case 0522
Student 0522 was interviewed two years after their project. They gave two reasons for not participating, the first related to the cost, “it's too much time consuming”, elaborated “it’s not time consuming in a sense that it's like time consuming, it's time consuming because you're in a sense, like you're in a special mode of mind. It's like you're in attention and wherever you go at that point in time you’re always thinking and it's like one of these things is like out of the reach.” The second reason related to the benefit, the student said, “it wouldn't like benefit me. There was no like potential benefit in like grading or anything else.” Asked if they meant it wasn’t marked, the student replied yes. Later in the interview, asked if they would have taken part with hindsight, the student replied, “the whole idea seems like an out of the way commercial thing, like if I get some advantage out of it, yes definitely I would be
willing, like if I get money definitely. Otherwise, yeah I might be if it holds some resume value or something like that? Yeah definitely. Otherwise I'm not too sure about that.” Asked if they would do it if it gave “about 5% extra” on their MSC grade the student responded, “definitely, definitely I will join it. (Laughs) Can you do it now?”

9.4.5 Case 0529
Another student (Case 0529) was asked if it was hard to tell the benefits of a tool, she suggested that “in the beginning if you can show them what it can do, then they can kind of weigh it”. This was the approach that had been used in the second cycle (in which she participated) and the main improvement over the first cycle.

9.5 Knowledge Retention & Maintainability
When asked if others would find it easy to reuse his code one student replied, “No, mainly because it was poorly commented due to time constraints near the end of the project” (Case 0417). Other students expressed similar views, for example Case 0405 who started documenting but noted how this “tailed off towards the deadline as the pressure to complete increased” (more on pg 171). Putting off documentation until the end of the project seems to be a common approach which SERE to at least some extent mitigated. By the later stages of a project, time constraints usually mean this documentation is not done at all. We believe this is partly due to the value of the comments (for the researcher) decreasing over time as well as the actual pressure of finishing. Case 0629 noted how SERE allowed students to “invest in cheap time to get expensive time back”. Knowledge retention through documentation was a key part of this.

Another student, when asked what he could do to make it easier for others to work with his code, replied “provide some sort of additional information” (Case 0406). This type of response was again common with many students suggesting that the key thing they could do to improve their project’s reuse potential would be additional documentation.

When speaking about the dOxygen output student 0526 said, “for me the main thing that you can immediately see is what calls what and things like that, a direct visual representation. It's worth doing for the final result. It's very nice to have something
that can support your work in this way.” This comment comes after struggling with inherited code where this was not immediately obvious.

In Case 0405 the student commented that “the clear benefit of this [SERE] is for future users of the code, and not myself (a selfish outlook, but that's just the way it is).” As already mentioned, a similar view was expressed by Student 0628 when he said “I'm not really concerned with what people do with the code after September the 8th am I?” and then called documentation “an altruistic professional” motivation. In a later interview, however, student 0628 said, “in terms of reuse there are definitely benefits [including] the way of specifying comments in terms of what this does, why this does and again that process”. The next thing the student said was “even going back myself and looking at that information has helped me understand what that a class is about and why it’s there and it [(the documentation)] is incomplete and it’s not perfect but it certainly helps.” (See pg 174 for more). Again we see both knowledge retention for the longer term and improved maintainability.

9.6 Systematic Approach

The need for systematic approach is a basic requirement of research, indeed the OECD definition of research is “creative work undertaken on a systematic basis in order to increase the stock of knowledge” (OECD 2002). This definition is used by the UK’s Higher Education Statistics Agency in determining what is and is not research within the UK context (HESA 2004). In this work we aim to increase the systematic nature of research without damaging its creative nature. This means enabling researchers to work more systematically, without moving into the realms of “managing them”.

The concern over management of research was summed up in our previous work by Prof Farr who said, “I think there's a bit of a trend to turn research into a process that can be managed like an industrial process and I think that is really antithetical to the nature of research...we still need to recognise that it is at its heart an creative and somewhat unpredictable process” (Oboler 2002). It is this creativity and adaptability of research that most differentiates it from the industrial process which we examined in section 3.1 (pg 40). One similarity is the repetitive nature of some tasks that take place within the research process, for these (and only these) it makes sense to draw a
comparison to industrial process improvement and find the approach that involves the
fewest steps and least effort – for example generating full class diagrams from the
code rather than producing them by hand. The Process Descriptors used in SERE
aimed to provide flexibility, limit the amount of overhead and still capturing extra
knowledge of relevance to research. These three aspects can be used by the researcher
to create a more systematic approach to their work.

Student 0628 explained: “when you’re coding it then it's all in your head. Maybe it
would help me later on, when I come back and look at it, fair enough. But at the time
it didn't really help me clarify the issues at that time. Indirectly though going through
your notes and my own afterwards systematically and applying them and resolving
them helped me do a better job. I wouldn’t have done it in quite such a systematic way
previously but I might have by chance made those changes anyway.”

In discussion about his journal, a Student 0625 noted that their approach wasn’t
particularly formal. They were asked if they thought software engineering had to be
formal. The student replied, “You'd know better than I. I only did one course, but I'd
say it needs to be relatively formal in that it needs to be documented, where as what I
did was basically jotting down any troubles I had that week and what I plan to do for
the next week. Just generally a plan really.” The student was informed that this met
the suggestions of SERE. A systematic approach does not need to be overly formal,
merely consistent.

SERE recommended that people put aside a little time each week for documentation.
One non-participant (Case 0625) explained what happens without this systematic
approach, noting that he had not a line of comment in his code, he was asked if he
would aim to comment more next time. The student replied, “I'd aim to, as I did this
time. It's always a matter of, well I'll finish this component first, then the next
component starts, and it's like well, I'll just finish this component. Then it’s the end of
the term, I'll do it all at the end, but then obviously dead lines start hitting you.” This
view was also reflected by Case 0417 who, though a participant, delayed putting time
into SERE activities like documentation and FTRs, and then eventually ran out of time
for them altogether. Like case 0625, they too felt their code suffered from insufficient
documentation and as a result would be hard for others to use.
The systematic nature of documentation in SERE meant those actively participating did not have the problems student 0625 encountered due to the advice about documenting regularly and the inclusion of FTRs where commenting was inspected and documentation gaps highlighted. These steps ensured students followed a more systematic approach. After looking through the SERE process descriptors, student 0625 said, “anything like technical reviews I’d tend to shy away from… anything that goes beyond helping into the realms of dictating how things should be I wouldn’t do.” This reflects the tension between a more systematic approach and increases management. While the tools can be made available, unless researchers decide themselves to become more systematic, the only other options are the status quo or increased management of research. The latter option is problematic, as it would impact on the creative nature of research. Improvements to increase the systematic nature of research must be led by the researcher or not at all.

9.7 Increased Efficiency
The aim of our approach was not to help an individual researcher finish faster. While we believe the other benefits already discussed (such as increased knowledge capture and improved reusability) would lead to efficiency gains in the longer term, in the short term the SERE approach aimed to introduce software engineering to improve the research process. This means investing additional time, but hopefully doing so effectively. SERE also tried to reduce “wasted effort”, for example recommending that students produce only high level architecture diagrams and then use dOxygen to generate complete design documents after the code was written. We’ve shown that other approaches like technical reviews can uncover problems early, allowing more efficient debugging. In the short term the cost of SERE participation (largely in terms of time) needs to be less than the benefits (measures as time savings + improvements in quality).

Student 0629 noted that they were able to work more efficiently as a result of SERE. They said their biggest problem was “too much research, not enough time... [which] led to a bit of a rush at the end, although the process model and code commenting helped better plan coding/write-up in light of the limited time. Asked about the time SERE takes compared to the time it saves they said, “I’d say it's close to neutral,
maybe a little on the positive side, maybe a little on the negative side, you get back what you put in. It's very close, I don't know if there is a slight imbalance either way.” When asked “Do you feel it shifts when you put in the time?” the student replied “Yes. We invest in cheap time to get expensive time back.” To put this in context, the student’s comment immediately preceding this was that “It's useful, it's helpful, it does involve extra time, but it does pay you back. It does pay back, it does save you time”. The apparent contradiction is because SERE saves “wasted time” (e.g. trying to remember how code worked) by spending time improving the quality (e.g. through documentation). These are both examples of costs and savings given by this student.

Asked about more open-ended projects, student 0629 said that without the deadline there wasn’t the same value for the time being saved at the end; however they added “I'd say the documentation guidelines in particular will save a few nasty afternoons staring at code and wondering what the hell it is doing. And assuming people weren't using something similar already, dOxygen comes in very handy.” The greater efficiency later on in the project life-cycle can be of benefit to other researchers on longer projects as it would enable papers and reports to be created in less time and with less effort. This can increase the chance of meeting a deadline at short notice, as well as increasing the quality of material produced under pressure.

The observation that participants greatly increased the amount of information they captured (compared to non-participants who often left documentation to the end and then ran out of time) and were generally able to participate, suggested that sufficient wasted time may have been released though SERE to negate the additional time cost of participation. The additional quality improvements for the same time invested can be seen as an efficiency gain.

9.8 Increased Reuse

Our previous work (Oboler 2002) has shown that longer term projects have more benefits including more publications, more spin-off projects, etc. One reason for this is that code can be reused, effectively moving the starting position in our Pipe Model of the research process (see Figure 2 pg 7). In this section we look at the SERE factors enabling reuse followed by a discussion on reuse from the case study data.
### 9.8.1 Reuse Enabling Factors

In the first year students were asked if their project could be extended by future MSc students. We have divided the students/projects into those where the students felt their work could be extended, see Table 23, and those where they could not, see Table 24. These tables have been previously presented (pg 122 and 123). Each row represents one student and whether they used other people’s code, which group they were in (the control group, experimental group, or neither i.e. a non-participant), whether the student felt it would be easy for others to use their code, and whether they used a high level diagram.

**Table 23 Projects where students felt their work could be extended**

<table>
<thead>
<tr>
<th>Others code</th>
<th>SERE participant</th>
<th>Easy to use your code</th>
<th>high level design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Experimental</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Control</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Neither</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Neither</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Table 24 Projects where students felt their work could not be extended**

<table>
<thead>
<tr>
<th>Others code</th>
<th>SERE participant</th>
<th>Easy to use your code</th>
<th>high level design</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Experimental</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Control</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Experimental</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Neither</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
There appears to be a relationship between projects that students felt can be extended and both the student’s reuse of other people’s code and the student’s production of a high level diagram. Some projects by their nature cannot be extended. This is the case with the fourth item in Table 24 where the student has done everything “right” but felt their project could not be extended. In their survey response the student explained that “as it was designed for demonstrating and evaluating existing concepts, I'm not sure how it could be extended for much else”. Another view on this data shows that of those who used a research journal (six students) all but one (the same student as above) felt their code could be extended. Of those not using a journal the results were evenly mixed. This could indicate the benefit of a journal, but we feel the use of a journal (if present) is rather indicative of the amount of effort students invested throughout the project development time on non-coding activities. It is we think these activities rather than the journal itself which increases the potential for reuse.

In Cycle Two only four surveys were returned and these were all from participants, all of whom felt their code could be reused, all of whom felt it would be easy to reuse their code, and all but one used a high level diagram. The one who did not use a high-level diagram is also the only one who did not use other people’s code. This data has already been shown in Table 8 (pg 124).

In the Third Cycle six surveys were returned, three from participants and three from non-participants. All except one student felt their work could be easily reused, and this student provided an extended comment noting that the difficulty would arise from a lack of documentation. Again the one student in this sample who did not use a high level diagram also did not reuse code. Only one other student did not reuse other people’s code and this student still used a high-level diagram. Only one student felt their work could not be extended and put this down to the nature of their project. The data was presented in Table 9 (pg 126).

In the next section we examined the students’ comments on reuse from their surveys and interviews. The first year survey answers showed concerns about the quality of the students’ code, and more specifically about the quality of documentation. This
shows a level of dissatisfaction with the final product which results from the rush the
meet the deadline. In the later years the dissatisfied set of students appear not to have
returned surveys, but comments about documentation making reuse difficult were still
observed through post project interviews.

9.8.2 Case Study Results on Reuse
In the post project surveys of a non-participant (Case 0625) the student was asked if
others would find it easy to work with their code. They replied, “Not at all - it's very
badly commented and would probably confuse people.” Further discussion revealed
there were no comments at all in the code. The student stressed that components of
their code could however be reused in a black box manner.

Student 0526 extended an open source product as their project. They had not looked at
the design of the open source product and it was suggested in an early meeting that
they do this before starting any of their own coding. Specifically they were advised to
generate a class diagram so they could focus their effort on relevant classes. The idea
of generating the diagram rather than creating it by hand was new to the student. They
had intended to use “sketches, UML or something” and it was suggested that they
consider a class diagram and interaction diagrams particularly to show how their work
interacts with the existing code. After the project the student was asked what benefit
they got from documentation, and they replied, “to be honest I didn't quite use it for
myself. But basically for people who might work on it I think it's going to be a lot of
help that I didn’t have, because the code I had had minimal comments. Very minimal
and it was quite hard for me. Now I think it's going to be quite easy for someone.”
When asked if someone else could reuse their work the student replied “yes definitely,
I think that is one basic use of dOxygen, to support the work and make it easier for
someone else. I feel it is a responsibility to document your work, otherwise it is more
or less worthless except on you, the other guy has to do reverse engineering.”

Student 0405 said that in the code they inherited the “documentation consists almost
entirely of high-level overviews of the architecture, meaning that its usage was largely
determined through studying example code. An API for key components would have
been useful. Code was mostly undocumented.” In this case the student was lucky as
the nature of their project meant they had allocated a significant amount of time to
study the code. This time gave them “a reasonable understanding of its operation”, although one aspect they wanted to test proved problematic. The student explained; “evaluating this is difficult without lower level documentation of the components to understand how they are to be used.”

In response to survey questions asking if their code could be reused, one student in the first cycle responded “I don't honestly think anyone else would be the slightest bit interested in it. Besides, I don't want the hassle of someone emailing me.” These aren’t particularly convincing or relevant answers, specially given that in this particular case their supervisor’s expert opinion is that the work was both reusable and highly likely to be reused. It does however highlight the human factors element once again. If researchers do not see a benefit for reuse some will actively obstruct it.

As previously discussed (pg 169), participation also increased the use of documentation and this in turn increases the potential for reuse.

### 9.9 Increased Satisfaction

Students’ satisfaction with SERE as a means of assisting them with their research increased as SERE became more mature. This was expected given the evolutionary nature of the development and refinement of SERE and is itself a positive outcome.

In the first year where the approach was mostly about providing tools, results were mixed. Two students who said they didn’t benefit commented that they weren’t particularly involved; these students were members of the control group so their non-participation was largely a result of the experimental design. One of these students (Case 0422) when asked if they benefited replied, “Not really, but the onus was on me I guess… I didn't get round to using it sadly”, another two replied “no” and commented that they weren’t very involved. One member of the experimental group replied similarly saying “not especially” and adding, “other than learning about the tools I didn't see any immediate benefit to the way I worked”. Student 0404, although in the control group, felt they had benefited, they said, “It was useful to know there are guidelines for coding and maintaining project work”. Student 0405 who was in the experimental group replied “Yes, particularly relating to code comments, which previously I always provided but in an unstructured way.” Another experimental
group participant, Student 0418 said “I benefited from the suggestions given by the project head (yourself). The code review was also useful for tidying up/commenting suggestions.” The mixed levels of satisfaction, particularly between the control and experimental groups led to the change in experimental design in the later two years of the experiment.

In the second year, of the seven participants to return their final feedback forms, six of them replied that they had benefited by being involved, the last replying, “I have not read your website”. Extended comments students provided on the benefits they felt they received, their overall impression, and the advice they would give new students offered the chance to participate, were very positive and have been previously present (pg 134). Comments for next year included “it's totally worth [it], and can help to understand the value of software engineering”, “do – it’s another line of support”, “Do it!”, and “Do so, and learn all you can about what to do, before the dissertation”. These sorts of comments show students who were very satisfied with their choice to participate and wished to share the experience with others.

In the third year only three participants returned their surveys, all three said they would follow SERE guidelines more closely with hindsight. A full discussion is available in section 7.3.3 (pg 135). All three said they had benefited. Asked to comment further during an interview Student 0529 said “you've seen the survey responses and as you can tell I was pretty mad happy with dOxygen”. Student 0528 comments also gave extensive feedback on the tools. When discussing the idea of capturing information using a process plan and then using the plan for discussion, they said that it “makes you feel warm and fuzzy inside, you know that you’re doing the right thing.” The student noted that they followed SERE “with alterations” but sometimes they would “feel guilty or I’d include something to make you happy even if I’m not sure about the benefit.” Asked if any of these extra things provided a benefit the student said “yes”, then revised this to a “maybe” noting that “I'm pretty sure, but it's hard to remember which bits I was unsure about.”
9.10 Threats to validity

A number of threats to validity, as well as approaches to mitigate these threats, have already been discussed during the course of this thesis. In this section we’ll discuss internal, external, and construct threats to the validity of this work.

9.10.1 Internal threats to validity

Internal validity threats question the conclusions drawn from the data (Carver, Voorhis et al. 2004). These are summarised in Table 25.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation / Rebuttal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student marks may not be measured consistently</td>
<td>The variety of markers, their independence from the SERE research, and the academic safe guards already in place mitigate against low quality in the marking process.</td>
</tr>
<tr>
<td>Low numbers of participants</td>
<td>For some aspects of the research sample sizes are small. The use of case studies to gather data from many sources adds depth to the research and mitigates against exceptions over contributing to interpretation. Sample sizes are also generally small in empirical software engineering, so while a risk, it is not a particular fault of this research.</td>
</tr>
<tr>
<td>Samples were self selecting and may exhibit sample bias</td>
<td>The use of the delta metric (showing personal improvement) somewhat mitigates against this. If the project marks themselves had been used it would be possible to argue that “better students” were participating. The results show that in one year participants had better coursework marks, while in another they had worse coursework marks… however they consistently improved on their own past marks by more when they participated.</td>
</tr>
<tr>
<td>Interpretation of recollections may colour results</td>
<td>Interviews were recorded and comments from students have been exactly quoted when used as supporting evidence. Additional information on the research subjects has been provided to add context. Key questions were after asked in both surveys and interviews to allow exploration and confirmation.</td>
</tr>
<tr>
<td>Danger</td>
<td>Interview data was transcribed and entered into the NVivo</td>
</tr>
</tbody>
</table>
experimenter bias in qualitative results software. Transcripts were coded according to themes (e.g. cost / benefits, problems encountered, tools, etc) and summary reports were generated from these. This two step process separated the analysis of the raw material from the interpretation. The approach used was discussed with Prof Nigel Fielding, an expert in the area.

Table 25 internal validity threats

<table>
<thead>
<tr>
<th>9.10.2 External threats to validity</th>
</tr>
</thead>
</table>

External validity threats question the applicability of the conclusions to other situations (Carver, Voorhis et al. 2004).

The most obvious threat to external validity arises from the use of MSc students as sample researchers. Even accepting that the SERE approach is of benefit in the MSc situation, one could question whether the results are applicable to PhD students. The reasons for this are summarised in Table 26.

Table 26 external validity threats

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation / Rebuttal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSc projects are shorter than PhD projects.</td>
<td>Many of the benefits of SERE (such as documentation, systematic approach etc) will increase with larger projects.</td>
</tr>
<tr>
<td>MSc projects are more structured that other research</td>
<td>This appears not to be the case based on the case study evidence collected</td>
</tr>
<tr>
<td>MSc projects have less collaboration than regular research</td>
<td>This means SERE should have added advantages (due to improved communication) with other types of research projects.</td>
</tr>
<tr>
<td>Lancaster MSc students may be a special case</td>
<td>As research is typically undertaken in an ad-hoc fashion, the freedom provided in the MSc program at Lancaster is typical of the field. The lack of existing research in the area makes it very unlikely other university have a better approach. Research at Lancaster is applied rather than theoretical, but SERE is not aimed at purely theoretical research.</td>
</tr>
</tbody>
</table>
These threats have been somewhat mitigated by my own use of many of the tools, as well as the introduction of some of the tools to other PhD students. Further work would however need to be done to better support the claim to generalisation.

### 9.10.3 Construct threats to validity
A construct threat is one where the experiment does not map correctly to the general case. Specifically it occurs when the concrete measures in the study do not match the theoretical constructs used in the hypothesis (Judd, Smith et al. 1991). A summary of construct validity threats is shown in Table 27.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Mitigation / Rebuttal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSc projects are not research</td>
<td>While not all MSc projects made a significant new contribution to knowledge, from the perspective of the research the knowledge is new. The process is therefore a research process.</td>
</tr>
<tr>
<td>Students marks do not model research quality</td>
<td>The marking of MSc projects was carried out by academics across the department. The academics (experts in the area) take into account the research product and process during marking.</td>
</tr>
</tbody>
</table>

Table 27 construct validity threats

### 9.10.4 Benefit of triangulation
Triangulation is a key component of the case study approach (Yin 1994) which involves the collection of many types of data related to a single unit of analysis. The case studies conducted in this research generally made use of direct observation, meetings (audio recorded), surveys, interviews, formal technical reviews, inspection of artefacts (i.e. diagrams, code, documentation, website and dissertations), and in some cases the use of third party observations.

Interviews for this work were recorded. The first interview was a structured interview and the later interviews with each student were semi-structured. Planned questions (particularly to the post project interview) were based on students’ involvement and their answers to surveys. Claims made in surveys could be asked again in the context of an interview, and where the answer was confirmed this could be explored in more
depth. Where different answers occurred further questions tended to result in qualifications on the first verbal response. Initial answers to verbal questions tended to display the student’s biases and preferences, where as further thought by the student brought out more evidence based answers drawing on their research experience.

Examples of planned confirmation include:

- Initial survey (in the lecture) with the electronic sign up form later on
- Initial survey compared with questions in initial meeting
- Final survey responses (all student) with interviews
- Final survey responses (participants) with interviews and artefact inspection
- Process plans with final interviews

Other triangulation occurred when open questions in interviews led to discussion at greater depth on information already collected in past interviews or surveys.

Some specific facts checked with triangulation between surveys and interviews:

- Use of various process descriptors by each participant
- Benefit of various process descriptors to the student
- Effort invested in SERE
- Benefit of participation

The use of process descriptors and the effort invested were further investigated using observations and technical reviews. Other triangulation was dependent on opportunity. These have been found throughout the thesis where multiple types of data are used to present and confirm or elaborate on a specific topic.

9.10.5 Validity summary

There are a number of threats to the validity of this research. Some threats, like the general applicability of SERE, will require future work before they can be properly addressed. Others have been addressed through a systematic approach to the experimentation. Many of the treats, such as small sample sizes, are inherent to empirical software engineering. This work has sought to use the largest possible sample size within practical limits and the case study approach, with triangulation between data sources, adds further depth and confidence to our analysis.
10 Conclusions

In this research SERE, an approach to improving the research process in the academic computer science environment, was developed and incrementally improved over three years as its effectiveness was investigated. This thesis demonstrates that our approach, which adapts software engineering to the needs of the research environment, is compatible with the nature of research and provides significant benefits both to the researcher and to other stakeholders.

Benefits we examined included:

- **Knowledge retention and maintainability**
  The ability to capture information related to the research process as well as the development process. Such knowledge may be of benefit to the current and future research when replicating, improving or adapting the experiment or code.

- **A more systematic approach to research**
  The integration of software engineering ideas, tools and concepts to structure the process of research and that of development. The research environment requires a mix between a creative and a systematic approach, the approaches examined are highly adaptable and have a low cost in order to make them suitable through a mix of structure and agility.

- **Improved reusability**
  The ability to reuse research code can provide significant time savings for future researchers and allow faster and deeper research. The approaches used facilitate improved reuse by improving the quality of both code and documentation.

- **Improved efficiency**
  Approaches to eliminate “wasted time” for “accidental” tasks have been examined and integrated. Approaches that would impact on a researchers creativity and direction of progress have been avoided.

- **Researchers’ perception of benefits from the components and the whole**
  Increased satisfaction is a benefit that can lead to improved productivity. Increased satisfaction can also signify a noticeable improvement of the overall research process.
• **Increased quality of the research product as measured by markers**
  Students using the SERE approach were shown to increase their level of improvement (compared to earlier work) when compared to non-participants.

We also examined the nature of computer science research and developed and improved Process Descriptors (tools, approaches and guidelines) that formed the components of our approach.

Process Descriptors we created included:

- **A process Model for Supporting Research**
  A meta model for examining how support for improving the research process is provided. These models can help further improve the process.

- **The Personal Process Model**
  A new approach to planning and organising the research process, that can be integrated with a researchers existing programming tools to provide a seamless integration between meta-level research planning and research coding.

- **Coding Guidelines**
  A set of guidelines for documenting research code. These guidelines explain how researchers can capture information that would be of benefit to themselves and future researchers, e.g. their rationale for design decisions.

- **Technical Reviews for Research**
  An adaptation of Formal Technical Reviews to the research environment, allowing researchers to quickly gather feedback from other experts who may not be involved in the project.

- **A reengineering pattern to aid us in supporting the researchers**
  To facilitate technical reviews a reengineering pattern was created that allow an outsider to contribute to a review after 2 hours preparation.

We also examined the use of high-level diagrams and other forms of documentation by researchers. We collected feedback on the open source dOxygen software that underpinned some of our process descriptors. Each of these items and each of the process descriptors have been examined in depth in this thesis. This work has shown
that the assumptions of our previous work, the RAISER / RESET SDLC on which we built this work, hold true, and that an approach based on these assumptions can be practically implemented. The SERE project, presented in this thesis, has significantly increased our understanding of the research environment and shown that tools can be created and adapted to meet these constraints.

As part of this work we have provided an abstract model and metric for discussing process improvement of research and the CARE framework relating the research environment to the CMM. The ideas along with the RAISER / RESET SDLC form a foundational which we hope will support future work in the area of process improvement for academic computer science. While our experimentation shows that software engineering and a process improvement approach can be used to improve the research environment for at least some classes of researchers, significant investment is needed to validate this with larger long-term projects and other classes of researchers.

In the following sections of this conclusion we discuss each of these issues in more depth, and then end with a discussion of future work in this area.

### 10.1 Benefits Demonstrated

In assessing the benefits of SERE we looked at both our volunteer participants and those who chose not to participate. Reflections from both groups are provided through our case studies. An analysis of students’ marks was conducted using the delta metric which we defined as the difference between a student’s average coursework mark and their project mark. Participants not only had a higher delta on average, but also had a lower standard deviation than occurred in the delta sample for non-participants. This was best demonstrated by a box plot of the combined results from the second and third year of our experiment in Figure 17 (pg 197). These results suggest our approach has the usual benefits of a quality improvement approach, namely increased average quality and increased consistency across the sample. A more detailed analysis of other benefits was also possible due to the case study approach.

Knowledge retention as a benefit of improved documentation was demonstrated through the SERE approach. In our previous work, knowledge retention was mostly considered in terms of external documentation the benefits of which were “mostly in
the area of knowledge transfer, that is communicating knowledge about the project between the research group developing it and other researchers” (Oboler 2002). In SERE we integrated the idea of knowledge retention with that of source code documentation. This occurred not only for the rationale behind the research, but also for the research process. Both these concepts are new to this research. The benefit of integrating knowledge capture with a researcher’s development environment led not only to improved documentation for future researchers, but also to improved knowledge retention over the life of the project. As knowledge was captured right through the process, rather than simply at the end, researchers found that documentation they made earlier reminded them of aspects they would otherwise have forgotten. Our previous work concluded with doubts that such benefits could occur using existing techniques. This work has shown that the new techniques we introduced can provide a benefit of knowledge during a project, a significant improvement over past attempts to use industrial style software engineering, or indeed no software engineering at all.

We have also noted benefits in maintainability, reusability and adaptability from participants in SERE. These are largely a result of improved documentation and improved design, which in turn were influenced by our coding guidelines and technical reviews. Research code is typically not built to be reused, yet more and more projects are reusing past code. In this thesis we have examined issues resulting from the reuse of existing code in research projects as well as steps researchers have taken to improve the reusability of their own code. High-level diagrams have been shown to be helpful for reuse, as is working on a project that reuses others code. Our coding guidelines provided a structure that allowed researchers to produce more readable code and to document their ideas in a way that would assist future researchers focus their effort better. Our Technical Reviews highlighted potential problems that would make reuse or understanding of the code by others more difficult and this lead to researchers improving their code.

While participants were free to use or not use any part of SERE, those who actively engaged, found that our approach gave them a more systematic approach to their research. The most obvious source of this is the Personal Process Model which allowed researchers to move from an ad-hoc approach (in the computing sense, see pg
29) to a more planned approach tailored to their own needs. The use of Coding Guidelines and the recommendation to document in notes while coding then perform a cleanup each week also added to a more systematic approach. Technical reviews highlighted inconsistencies in approach and gaps in documentation. The various approaches interlinked, the coding guidelines introduced the dOxygen tool, the Personal Process Model extended the use of the tool, and the Technical Reviews were based on researchers’ efforts with both the documentation guidelines and the Personal Process Model. Other aspects like High Level diagrams were also examined in reviews. As students became more involved their approach became more systematic and while personalised, the use of standard Process Descriptors also caused some aspects of the work to become more uniform across the sample. In many cases common practices simply reflect the adoption of what became accepted as best practices, for example, students uploading dOxygen documentation of their project to their website.

One of the most obvious benefits to be expected of a process improvement approach is improved efficiency. We discussed this as one form of improvement in the introduction. While SERE has helped reduce wasted time, on average the impression from students is that it neither saved nor cost additional time. Given the SERE causes students to undertake additional software engineering work (the benefits of which have already been discussed) this shows not only that wasted effort was reduced, but that the effort was then spent on improving the research product and process in other ways, for example through improved documentation. In some sense the improved process has functioned as insurance, the additional effort not adding limited benefit to the researcher but helping to prevent various types of disasters, e.g. the researcher forgetting ideas, not catching bugs till later in the process, or losing files. The nature of the additional work as value added rather than wasted effort is itself a form of improved efficiency and a worthwhile benefit of the SERE approach.

A major thread of this research was the evaluation of the overall cost and benefit of the SERE approach and of participation. Due to the difficulty of measurement and the different value researchers place on both costs and benefits we have relied on the researchers’ perception of benefits and costs and gathered this data from both surveys and interviews. In the case of participants this was gathered repeatedly at various
points in the research process. The results have shown that overall participants were happy with SERE and felt they had benefited. Those who felt they hadn’t benefited were those who also admitted to not actively participating. Participants generally felt they would participate more if given the opportunity again. This all suggests a very positive outcome from the researchers’ perspective, though this is not surprising given that we’d expect participants to opt out of any approach that was not working for them. The increased take up in Technical Reviews along with the positive feedback from those who participated, demonstrates how some of our approaches improved over time, leading to increased adoption. The overall adoption of SERE has also been discussed in this thesis and provides a sanity check on the suitability of our approaches for the research environment, and demonstrates the benefit people saw in participation. The individual components of SERE were also evaluated and this is presented in the next section.

10.2 Process Descriptors Created and Evaluated
Our SERE approach used many different tools, approaches and guidelines, which we collectively refer to as Process Descriptors, to assist students in improving their research process. In Chapter 7 (pg 113) we presented the results of the experiment which through surveys and case studies referred to some of these Process Descriptors and their effect on the research and the researcher. Our surveys asked participants to rank the various Process Descriptors in terms of their usefulness and through interviews we captured information on the costs and benefits researchers associated with various Process Descriptors. In Chapter 8 (pg 160) we discussed a few of the more novel Process Descriptors in depth. The remained of this section summarised our conclusions on these approaches.

10.2.1 The Process Model for Supporting Research
The Process Model for Supporting Research (pg 160) was developed in stages and reflected not only the development of our own understanding, but also steps in the evolution of a more mature systems model of the research environment. The model links to the CARE framework that we introduced as a translation of CMM to the research environment in Chapter 2 (pg 63). In the Process Model for Supporting Research the role of the researcher and software engineer in relation to the development of an improved process and of improved process descriptors is mapped out. In this work we were not merely conducting an experiment, but using an in vivo
approach to model the inclusion of software engineering support in the research environment. One important realisation was that the process of research, like the process of learning, can only be improved by the student. Like a teacher who seeks to facilitate improved learning, so must a software engineer in the research environment seek to facilitate process improvement. This realisation led directly to the development of the Personal Process Model. The Process Model for Supporting Research has been published as “Reflection: Improving research through knowledge transfer” (Oboler, Sommerville et al. 2006).

10.2.2 The Personal Process Model
The Personal Process Model (pg 162) provided the tool for researchers to take control of their own research process. Based on the idea of process programming (pg 55), the Personal Process Model integrated the documentation of process related ideas, actions and notes, with the development of the research product. Documented using an Object-Oriented approach, the Personal Process Model allowed us to build on researchers’ skills at abstraction and decomposition to structure their process in a meaningful way. The approach allowed a wider view of the research and moved research planning to a more strategic level rather than simply a concern about the next step. Feedback from those who used the approach was very positive with one commenting that it gave “Clarity, also in terms of neatness and presentation” and another saying it was “very handy and I got quite a lot out of it”. The Personal Process Model as a concept has great potential for process improvement within the Computer Science research environment and provides one of the most significant contributions of this thesis.

10.2.3 Coding Guidelines
One of the earliest Process Descriptors we introduced was the Coding Guidelines (pg 172). Originally introduced as a documentation standard, encouraging a less onerous approach caused an increase in adoption and satisfaction. The guidelines introduced researchers to dOxygen and recommended different types of comments that would record the sort of information researchers are actually interested in. As was explained to participants, knowing that a “for loop” is a type of loop is a worthless comment at this level. Conversely, knowing whether a search algorithm was the first thing the researcher thought of, or the result of a detailed evaluation or experimentation, is very valuable information and not otherwise reflected in the code. The Coding Guidelines
focused documentation on researchers’ rationale during coding and documentation of their effort expended and other research paths that may or may not have been tried. The documentation was reviewed as part of the Formal Technical Reviews. A paper on the coding guidelines has been published as “Research Documentation Guidelines: Capturing knowledge, improving research” (Oboler and Sommerville 2007).

10.2.4 Technical Reviews for Research
Both the Coding Guidelines and Personal Process Model were inspected through Technical Reviews (pg 176). Technical reviews are widely used in industry and can be seen as analogous to a viva situation, but can, as we showed in this work, be widely used in academia to share information, improve research approaches and increase collaboration. Reviews allow experts to become involved in an effective way that does not require a large amount of investment from either the researcher or the reviewers. The direct review of research processes and early products (code, plans etc) can significantly improve the research process while providing researchers with a fresh view of their work. A paper on Technical Reviews was presented as “Formal Technical Reviews for Research Projects” (Oboler 2007).

10.2.5 Reengineering Pattern
In preparing for the technical reviews we used a reengineering pattern (pg 260) that limited the preparation time to two hours. This allowed us to focus and support the researchers without becoming lost in the detail or making the review so large as to be unmanageable. The reengineering pattern was based on two existing patterns and involved an inspection of code and dOxygen generated documentation. Such an approach would be essential to a large scale software engineering lab with many projects to support.

10.2.6 High-Level Diagrams
High-level diagrams (pg 166) and other forms of documentation used by researchers (pg 169) were also examined in more depth in this thesis. Other process descriptors used but not presented in depth include: versioning / synchronisation tools, support Process Descriptors such as the installation and setup guide for dOxygen, various diagramming and drawing tools, a poster with a quick reference guide to SERE, a getting started guide, a research notebook template, and preparation guidelines for the
technical review. Participants’ views on these tools can be seen in the survey results and case study introductions presented in chapter 7 (pg 113).

10.3 Foundations Extended
The Research Process Optimization Framework that we presented in the introduction to this work (pg 5), and published as (Oboler, Lock et al. 2007), represents a new abstract way of thinking about process improvement in the research environment. This understanding at a project or research group level combined with engagement of the Process Model for Supporting Research (pg 160) provides researchers with a foundation to improve the RAISER side of a RAISER / RESET SDLC. It can also allow departmental progression through our CARE framework (pg 63) and improve the capacity of research of a department. These new developments extend the foundational work of the RAISER / RESET SDLC and have the potential to facilitate systemic improvement.

Prior to this work, RAISER provided a set of guidelines recommending that software engineering for research to be “reactive”, changing as the research changes, and “assistive” giving benefit to the researcher and avoiding any tools or approaches that would be a burden to the research. These two constraints formed a key part of the RAISER approach and were given high priority when selecting approaches. In this work we have shown that a package of tools can be created that meets this specification. We have also shown, at least within the context of MSc projects, that the constraints are indeed valid and, if anything, stronger than we had expected. While this may be a result of the short project times experienced by MSc students, other research projects also often work under tight time pressures. SERE added additional criteria for adoption, namely low evaluation cost and a low barrier to adoption.

This work demonstrates that the RAISER part of the RAISER/RESET SDLC can be implemented in practice. It has also gone some way (through technical reviews and our engineering pattern) in creating and testing a framework for RESET interaction with the RAISER process. It was unfortunate that we were unable to test a full RAISER/RESET life cycle, but this process is too high risk and requires a greater window of time than we had available in this project.
10.4 Final Remarks and Future Work

In this thesis we examined “whether a RAISER-based approach can improve the research process in computer science academia”. We implemented the SERE approach and offered it to MSc Students over a three year period. The improvements and benefits we noticed suggest that our null hypothesis, i.e. that such an approach would make no difference to the research process was rejected.

Our experiments were based on MSc students; however we believe (as do many participants) that the approach would be equally useful to PhD students and other researchers. Further research would need to be carried out to validate this assumption. From our own experience, we have tried many of the tools and found them helpful in the course of our work. Now that the tools and approach have been developed, using them on PhD projects or with a larger research project would be the next obvious step.

The variety of areas of computer science research that the participants worked in show that the SERE has applicability for most areas of computer science research, though less so for non-coding theoretical areas. SERE may also meet the requirements of research and development departments in industry, and may provide a middle road between extreme programming on the one hand and the more traditional designed approach on the other. This would require further experimentation, preferably in an industrial setting. The one industrial based project that was included in our sample (by luck rather than by design) showed that the SERE approach can be implemented in an industrial setting and in this case certainly did no harm.

It is important to note that SERE presented one framework and set of process descriptors for the implementation of RAISER and other implementations with different process descriptors are possible. If the process descriptors are managed across a smaller range of research areas, or ideally within a research group where specialist tools, approaches and knowledge can be shared, this is likely to provide greater benefits than the more generic approaches and tools we have used here. We hope this work will encourage such future exploration.

The Personal Process Model was developed late in our research process and only used in the third cycle of the experiment. Although the results are very positive, the sample
is unfortunately small. Further examination of the Personal Process Model, especially by multiple researchers within a research group, would be the next logical step.

While the level of documentation in participants’ projects has been improved, the impact this has on the RESET task and on future work has not been investigated. Such an investigation would provide valuable insight into the longer-term benefits of the SERE approach.

The CARE Framework presented in this thesis is only an outline. A more detailed elaboration based on a field study of the current state of departments would provide a valuable addition to the further development of this framework.

Though unavoidable, the evolutionary approach in this case started from scratch. Had we begun with the process descriptors that now exist progress would have been faster and the results more consistent. While greater breadth has been achieved through a mixture of methods (for example using a control and experimental group in the first year), this has limited the size of the sample that used specific process descriptors and made some worthwhile comparison difficult. Though the sample sizes we have relied upon are considered reasonable for software engineering work, the standard in empirical software engineering is lower than for empirical work in other scientific areas. Larger samples, and preferably from multiple universities, would provide more confident results. While not feasible in this project, it remains desirable to evaluate the SERE approach.

Understanding that researchers must be responsible for their own process, and that engineers may only assist researchers, proved a critical break through in understanding. The relationship is similar to a teacher who can assist a student with their learning, but can not do the learning for the student. This realisation means that an improvement in the research process must be the result of a culture change and the impact of tools and other process descriptors is highest when they facilitate this change. Had such a realisation been reached earlier, the Personal Process Model may have been invented earlier and the impact of SERE may have been more focussed and more significant.
While there is much work still to be done in the area of process improvement in the research environment, the SERE approach presented in this thesis offers significant improvements over the existing ad hoc approach used by most academics. We hope this work leads to further improvements, better adoptions of these improvements and better and faster research in the field of computer science.
11 Appendix 1: Surveys

11.1 About the Surveys

Each year two surveys were given to all students, a third survey was given to participants.

The first survey is given out before students start their project. It aims to get an idea of the background and preconceptions of the MSC students. There are two version of this survey. In the first year it was released after the initial presentation. There was some difficulty getting them back, but many were in the end returned. In later years it was given to students during the introduction session and completed during the session. The form used in the later two years collects additional information, mostly about the students’ perceptions. In the later two years the first few questions on the survey (one to three) were shown in the slides, the form itself did not give any hint of the questions. This design was to prevent the early answers being influenced by the later questions and the direction of the questioning. These questions from the slides have been added to the survey as presented here.

The second survey given to all students was send out via e-mail each year just before the students were due to submit, with a reminder after the submission date. This survey collects data on the students experience. In the final year interviews were carried out in addition to this survey, usually immediately after the students demo. The interviews examined issues addressed in this survey in more depth.

The additional survey for participants evaluates SERE itself, the students final perceptions of it, and act as a check to collect data on how much of SERE students actually used. Items related to this survey were also examined in interviews with participants.

In the final year a very short six question survey was sent out to student some weeks before the main surveys. This collected some critical information.
11.2 Introduction Survey

11.2.1 MSc Introduction Survey 2003-2004

Dear M.Sc. Students,

As discussed previously my research requires some survey responses from all of you before you start your projects and again after the projects are over. This is to validate that the sample of people taking part in the SERE experiment are representative of the whole group, and to bench mark the SERE participants project experience against the class.

I really do need a reply from each and every one of you. These replies are NOT anonymous as I need to compare information from different sources and tie it to students. No student’s name will released or published. The factors being looked at are things like your past degree(s), age, sex, work experience, where you come from, project marks, etc.

To save you time below is a greatly reduced list of questions and with your permission I’ll chase up the other information from existing university records.

Name: __________________________________________

Your Software Engineering Experience / Training (describe):

__________________________________________________________________________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

Have you worked in Computing, IT or related jobs?  YES / NO
If yes, please say what you did and for how long: __________________________

__________________________________________________________________________________________

__________________________________________________________________________________________

Why did you / didn’t you choose to participate in the SERE experiment?

__________________________________________________________________________________________

__________________________________________________________________________________________

Did your friends play a part (encouraging or discouraging you): Yes / No

I hereby give permission for Andre Oboler to access my university records, including application records and grades for the purpose of the SERE research project. All data will be anonymised for future use.
Signed: __________________________ Date: __________________________

Please return this form to Trish or to me this week (i.e. Friday the 19th of March)
Questions? Please e-mail me: oboler@comp.lancs.ac.uk.
11.2.2 MSc Introduction Survey 2004-2006

Name: ________________________________
E-mail: ______________________________

Question 1: Software engineering

a) What is Software Engineering?
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

b) Who is it for?
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

c) What is the result of using it?
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

Question 2: Research

a) What do you think the problems are in building research software? (e.g. MSc projects)
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

b) Do you think software engineering can help? Yes / No
________________________________________________________________________

c) Why / why not?
________________________________________________________________________

Q3 Discuss with the person next to you what problems you think might arise if no software engineering is used in research projects. Write down 5 potential problems you discuss.
1) ________________________________
2) ________________________________
3) ________________________________
Q4 To gain access to SERE resources you need to fill this bit in, and sign below giving permission to gather other data e.g. nationality, grades, etc.

Your Software Engineering Experience / Training (describe):

Have you worked in Computing, IT or related jobs?  YES / NO
If yes, please say what you did and for how long:  

Why did you / didn’t you choose to participate in the SERE experiment?

Did your friends play a part (encouraging or discouraging you): Yes / No

Your non Lancaster e-mail address (to contact you once you complete) is:

I hereby give permission for Andre Oboler to access my university records, including application records and grades for the purpose of the SERE research project. All data will be anonymised for future use.

Signed: ___________________________ Date: ___________________________

Q5 What questions do you have?
11.3 Final Surveys

11.3.1 MSc Final Survey All Students 2003-2004

Section 1 (Admin only) - in case anything needs following up
Name:
Non-Lancaster e-mail address:
(This will be used if any of your answers need clarifying!)

Section 2 - Project experience
1. Did you use other people's code?
   1a. How would you describe that code? (e.g. open source project, supervisor wrote it, PhD student's code, other student projects, etc)

   1b. Describe any problems you encountered learning to use / integrating with this code, as well as the impact this had on your project (e.g. how much time did it take up, did you decide not to use part of it, etc)

   1c. Describe any benefits you got from using this code (e.g. how much time did you save, were you able to do any additional things etc)

2. Did you lose any work at any point during the project?

3. Describe any other problems you had during your project (and any solutions to these!)

Section 3 - The project's life
1. Do you feel your project can be extended or used for other projects?

2. Do you think someone would find it easy to work with your code (why/why not?)

3. If you could do one thing to make working with your code better for other people, what would it be?

Section 4 - Project Methodology
1. Did you keep a project journal (yes / no)

2. Did you use source control tools (e.g. CVS, sourceForge, SourceSafe etc)

3. Did you produce any form of high level design before starting work coding?

4. What sort of documentation did you create for your project?

5. If you had to do it again, would you use more or less software engineering?
Section 5 - SERE
1. If you didn't sign up:
1a. What impression of the project did you get from other people?
1b. Given the choice again, would you have signed up?
1c. Did you see any of the SERE documentation? (if so what did you think of it?)
11.3.2 Additional Final Survey for Participants 2003-2004

Section 1. Admin
Name:

Section 2. Time
1. Did participating take more, less or about the time you expected it would?
2. Would you have preferred more contact time (lectures, training, meetings) or less?
3. Overall, did you feel RAISER encouraged you to do more or less engineering than you would otherwise have done?

Section 3. Impact
1. How much impact did the SERE guidelines have on the way you tackled your project?
   [a. I didn't read them] [b. None] [c. Some impact] [d. a lot]
2. Looking back, would you followed them more or less closely if you did it again?
3. What was the most useful part of SERE?
   (Rate the following 1=useless, 2=some use, 3=useful, 4=very useful, 5= the most useful bit!)
   3a. Getting started guide and general guidelines some use
   3b. journal advise useless
   3c. tools web page useless
   3d. CVS useless
   3e. coding guidelines useful
   3f. meetings with supervisor (Experimental group only) useless
   3g. Code review (Experimental group only) useless
4. Did SERE participation help in any way with the writing up? (Describe if yes)

Section 4. Tools
1. What tools (from SERE website or not) did you find particularly useful?
2. Would you recommend against any tools? (which ones and why?)

Section 5. Impression of SERE and RAISER
1. Do you feel you benefited from being involved? (how?)
2. What else would you have liked to see included?
3. What would you like to see changed?
4. What was your overall impression of how SERE was introduced and run?
5. Would you use any aspects of RAISER for other projects? (Which parts?) (Would they have to be research projects?)

6. What advice would you give to next years students when they think about getting involved?

7. As explained at the start, RAISER aims to give "maximum benefit, with minimum effort, now". How well does it do this (explain!)?

Section 6. RESET

1. How likely is it that your code can be of use to other projects? (1=no way - 10=certain)

2. Would you recommend your project be RESET?

3. If your code is RESET, would you like a copy of the RESET version?
11.3.3 MSc Final Survey All Students 2004-2005

Section 1 (Admin only)
1.1 Name:
1.2 Non-Lancaster e-mail address:
(This will be used if any of your answers need clarifying!)

Section 2 - Project experience
2.1 Did you use other people's code?
2.2 Did you lose any work at any point during the project?
2.3 Describe any other problems you had during your project (and any solutions to these!)

Section 3 - The project's life
3.1 Do you feel your project can be extended or used for other projects?
3.2 Do you think someone would find it easy to work with your code (why/why not?)
3.3 If you could do one thing to make working with your code better for other people, what would it be?

Section 4 - Project Methodology
4.1 Did you keep a project journal / diary (yes / no)
4.2 Did you use source control tools (e.g. CVS, sourceForge, SourceSafe etc)
4.3 Did you produce any form of high level design before starting work coding?
4.4 What sort of documentation did you create for your project?
4.5 If you had to do it again, would you use more or less software engineering?
4.6 Did you have a "group" of friends that supported you through the project and who you could talk to about problems?

Section 5 - Participation
If you are a SERE participant, ignore this section.

If you didn't sign up to SERE:
5.1 What impression of the project did you get from other people?
5.2 Given the choice again, would you have signed up?
5.3 Did you see any of the SERE documentation or hear about it from people? (if so what did you think of it?)

Thanks for taking the time to fill this out, and to those who have had or will be having interviews.

Andre Oboler
SERE
PhD Candidate
11.3.4 Additional Final Survey for Participants 2004-2005

Section 1. Admin
1.1 Name:
1.2 Non Lancaster e-mail address:

Section 2. Time
2.1 Did participating take more, less or about the time you expected it would?
2.2 Would you have preferred more contact time (lectures, training, meetings) or less?
3.3 Overall, did you feel SERE encouraged you to do more or less engineering than you would otherwise have done?

Section 3. Impact
3.1 How much impact did the SERE guidelines have on the way you tackled your project?
   [a. I didn't read them] [b. None] [c. Some impact] [d. a lot]

3.2 Looking back, would you followed them more or less closely if you did it again?

3.3 What was the most useful part of SERE? (rate the following 1=useless, 2=some use, 3=useful, 4=very useful, 5= the most useful bit!)
   3a. Initial lecture
   3b. Getting started guide and general guidelines
   3c. journal advise + sample journal
   3d. tools web page
   3e. CVS
   3f. Allways Sync
   3g. Dia
   3h. GIMP
   3i. dOxygen
   3j. coding guidelines
   3k. Initial meeting with me (April / May)
   3l. Technical review meeting (July / August)
   3m. Other meetings and e-mails with/from me
   3n. dOxygen sample code and diagrams
   3o. dOxygen installation and setup instructions
   3p. Technical review preperation document

3.4 Did SERE participation help in any way with the writing up? (Describe if yes)

Section 4. Tools
4.1 What tools (from SERE website or not) did you find particularly useful?

4.2 Would you recommend against any tools? (which ones and why?)

Section 5. Impression of SERE
5.1 Do you feel you benefited from being involved? (how?)
5.2 What else would you have liked to see included?
Chapter 11

Appendix 1: Surveys

5.3 What would you like to see changed?
5.4 What was your overall impression of how SERE was introduced and run?
5.5 Would you use any aspects of RAISER for other projects? (Which parts?) (Would they have to be research projects?)
5.6 What advice would you give to next years students when they think about getting involved?
5.7 As explained at the start, the RAISER model SERE implemented aims to give "maximum benefit, with minimum effort, now". How well does it do this (explain!)?

**Section 6. REUSE**

6.1 How likely is it that your code can be of use to other projects? (1=no way - 10=certaint)

6.2 Would you recommend your project be RESET?

Thanks again for your participation and help with this research. Any additional thoughts, advise or comments would be most welcome.

Thank you for your participation throughout the year, and particularly now as things come to a close.
11.3.5 MSc Six Quick Questions 2005-2006

Your Name:

1) What is the biggest problem you've faced with your project?

2) Is there anything that caused you to waste time? (if so what?)

3) What is the most useful tool / piece of advice you were given?

4) Do you have a diagram of your system? (if so, how detailed is it?)

5) Describe how you approached the project. This you could mention include:
   - did you plan how you'd do things at the start
   - did you plan before doing each step
   - did you make lists
   - did you discuss the project with people (what sort of people?)
   - did you do more planning, coding or reading
   - how useful was your initial schedule?
   - what else had an impact on how you did your project?
   - what sort of impact did it have (what did you do different, was this good or bad?)?

But you do NOT need to mention all these and may want to talk about other related things that affected the way you did your project.

6) Describe one thing that would make doing your research easier:
Chapter 11

11.3.6 MSc Final Survey All Students 2005-2006

Section 1 (Admin only)
1.1 Name:
1.2 Non-Lancaster e-mail address:
(This will be used if any of your answers need clarifying!)
(Unless you indicate not to, this address will also be given to Trish so
she can track you down in 12 months time to get info about how your
careers are going)

Section 2 - Project experience
2.1 Did you use other people's code?
2.2 Did you lose any work at any point during the project?
2.3 Describe any other problems you had during your project (and any solutions to
these!)

Section 3 - The project's life
3.1 Do you feel your project can be extended or used for other projects?
3.2 Do you think someone would find it easy to work with your code (why/why not?)
3.3 If you could do one thing to make working with your code better for other people,
what would it be?

Section 4 - Project Methodology
4.1 Did you keep a project journal / diary (yes / no)
4.2 Did you use source control tools (e.g. CVS, sourceForge, SourceSafe etc)
4.3 Did you produce any form of high level design BEFORE starting work coding?
4.4 What sort of documentation did you create for your project?
4.5 If you had to do it again, would you use more or less software engineering?
4.6 Did you have a "group" of friends that supported you through the project and who
you could talk to about problems?
4.7 Do you think software engineering should have a role in research projects and why
do or don't you feel this way? (comments relating this question to your own
experience are welcome)

Section 5 - Participation
If you are a SERE participant ignore this section.

Clarification: You are a participant if you had access to the website and
used (or considered using) at least one thing from it, be that a tool, a
document or an idea. If you did this skip this section.

If you _didn't_ sign up to SERE:
5.1 What impression of the project did you get from other people?
5.2 Given the choice again, would you have signed up?
5.3 Did you see any of the SERE documentation or hear about it from people? (if so
what did you think of it?)
Thanks for taking the time to fill this out, and please let me know about interviews. They will only take 15 minutes and I want to speak to every one.

Best of luck with your MSc marks, and with your future careers!
11.3.7 Additional Final Survey for Participants 2005-2006

Section 1. Admin
1.1 Name:

Section 2. Time
2.1 Did participating take more time, less time or about the time you expected it would? (Please answer MORE, THE SAME or LESS)
2.2 Would you have preferred more contact time (lectures, training, meetings) or less? (MORE or LESS) (Also comment if you wish)
3.3 Overall, did you feel SERE encouraged you to do more or less engineering than you would otherwise have done? (MORE or LESS, add a comment as well if you like)

Section 3. Impact
3.1 How much impact did the SERE guidelines have on the way you tackled your project?
[a. I didn't read them] [b. None] [c. Some impact] [d. a lot]
3.2 Looking back, would you have followed them more or less closely if you did it again? (MORE or LESS - add a comment as well if you like)
3.3 What was the most useful part of SERE? (rate the following 0=don't know, 1=useless, 2=some use, 3=useful, 4=very useful, 5= the most useful bit!)
   3a. Initial lecture
   3b. Getting started guide and general guidelines
   3c. journal advise + sample journal
   3d. tools web page
   3e. CVS
   3f. Allways Sync
   3g. Dia
   3h. GIMP
   3i. dOxygen
   3j. coding guidelines
   3k. Initial meeting with me (April / May)
   3l. Technical review meeting (June/July)
   3m. The Process Model idea
   3n. The "Generic Template.java" to impelment the process model
   3o. Technical review meeting 2 (July/August) (looked at process model)
   3p. Other meetings and e-mails with/from me
   3q. dOxygen sample code and diagrams
   3r. dOxygen installation and setup instructions
   3s. Technical review preparation document
3.4 Did SERE participation help in any way with the writing up? (Describe if yes)

Section 4. Tools
4.1 What tools (from SERE website or not) did you find particularly useful?
4.2 Would you recommend against any tools? (which ones and why?)

Section 5. Impression of SERE
5.1 Do you feel you benefited from being involved? (how?)
5.2 What else would you have liked to see included?
5.3 What would you like to see changed?
5.4 What was your overall impression of how SERE was introduced and run?
5.5 Would you use any aspects of RAISER for other projects? (Which parts?) (Would they have to be research projects?)
5.6 Do you think SERE should be included as part of the MSC course for next year? (Note that the project as an experiment ends this year, but similar support based on SERE could be offered to support the project).
5.7 If SERE (or something similar) was offered voluntarily, what advice would you give to next years students when they think about getting involved?
5.8 As explained at the start, the RAISER model (software development lifecycle) which SERE implemented aims to give "maximum benefit, with minimum effort, now". How well does it do this? (explain why you feel this way)

Section 6. REUSE
6.1 How likely is it that your code can be of use to other projects? (1=no way - 0=certain) Why?

6.2 Would you recommend your project be RESET? (i.e. profeshionally cleaned up, redocumented, restructured if needed etc)

Thanks again for your participation and help with this research. Any additional thoughts, advise or comments would be most welcome. (You can add them here)

Thank you for your participation through out the year, and particularly now as things come to a close.
11.3.8 Supervisor Survey 2003-2004

Admin
1. Your Name:
2. Students Name:
3. Project name:

Questions - As far you as know...
1. Did the student reused other people's code?
   Yes / No
1a. Was any of this code from or the result of other research projects?
   Yes / No
1b. Did the student mention any difficulties in working with the code?
   Yes/ No
   Comment if you wish:

2. Did the student lose work at any point?
   Yes / No

3. Do you feel the project can be extended or reused for other projects?
   Yes / No

4. How likely is it that this projects code can be of use to other projects? (1 = no way through to 10 = certain)
   1  2  3  4  5  6  7  8  9  10

A number of the MSc projects will be re-engineered as part of my PhD work. Resources to do this are very limited. Do you feel this project would be a suitable candidate for inclusion, and will you be offering an MSc project this academic year that could benefit?
12 Appendix 2: Process Descriptors

12.1 The Getting Started Guide

12.1.1 About the getting started guide
The getting started guide was introduced to students via e-mail when they signed up. This was the first process descriptor they came into contact with and it introduces some of the other process descriptors. The release of the getting started guide was timed to occur well before students have seriously started thinking about their projects.

12.1.2 Starting your research project (The guide)

The nature of research
Building research software is usually very different from building a product in industry or as part of a student project. While the area you undertake research in is usually fixed, the result and methods used are often flexible. A research idea is not guaranteed to work, you may have to backtrack or try other approaches. Even when an idea does work, you may come up with a better idea and wish to try that as well. These things all mean research cannot usually be planned. This does not mean you cannot plan your time, just that you are unlikely to have “finished” when your development time runs out. You may not have all the features you wanted or you may feel you have used an inefficient algorithm or approach. The goal of research is not simply to build something. Without a write-up the product itself is, from a research perspective, worthless. Without keeping track of your method and the source of your ideas writing up is extremely difficult. The better your records as you go along the easier your write-up will be.

Planning to work

Initial Research Statement
Before you begin you need to know what it is you intend doing. This should be a broad statement that covers many (but perhaps not all) of the possibilities that may occur as your research progresses. In the case of MSc projects, this has been done for you by your supervisors.
Over the duration of the project you will need to expand upon the project description, narrow down what aspects you will tackle and what methods you will try. After you have done some background reading (probably provided or recommended by your supervisor) you should have your first draft of this research plan. You should also have a clearer idea of what questions you need answered and what further background reading you need to do.

Literature review
As you read, make notes of what has already been tried. As you form an idea of your own approach, note how it differs from past approaches. This might be by using a new
method to solve the problem or by tackling a different aspect of the problem. Refine your research to include these details.

Your literature review should justify your research plan by:
- Defining the problem as others have seen it
- Discussing the results of past research
- Define the gap in the research (what has not been done, has anyone said it has not been done?)
- Relating your problem to other problems
- Discussing the approaches taken to solve this or similar problems in the past
- Provide background on the method you intend applying to your problem

The Research Plan
Your research plan should then:
- Explain your approach (the justification should be in your literature review)
- Provide a time table including allowing time for writing up

Note: this plan is a live document. You should change it as new ideas occur to you… but make sure to note your changes and what inspires them.

While you work

References
While you do your research it is important you keep track of any sources you read, especially if they give you new ideas.

Keep track of ideas you have and who they came from / what inspired them:
- what you read / who you spoke to that suggested it
- what you read / who you spoke to and what about that gave you the idea
- Analogies to other areas of research are very useful, but keep track of the sources that lead you to that area as well as the sources you looked at to further explore that area.

Some of this case be added to your literature review, other can not but will should be noted in a research notebook.

The research notebook (RAISER 1)
A research notebook can be a physical book or an electronic document. It will for most people become a combination of both. However you store the information, it is important you keep a backup.

The notebook is a personal workspace. Documents you create for your supervisor or for yourself should all be stored in the notebook. In addition you should add notes from your meetings with your supervisor and discussion with your peers. Anything that is likely to be of use when writing up should go in the note book. You should also add your own questions. As you progress with your work you can come back to these and add references to other parts of your notebook where the answers have been
recorded. A word version of the notebook is available on the SERE website. It is intended that a web version will be made available at some point.

RAISER 2: Once you have read sufficient articles find 1 journal and 2 upcoming conferences you could potentially submit your own work to. Make a note of these in your research notebook. (You may wish to add others.)

**Keeping the writing up in mind**

It is important to think about writing up right through the process. In your report you will eventually need to include:

**Problem statement**  
from the initial research statement, refined with information from the literature review and the research plan

**Past work**  
From the literature review – you should mostly be looking to cut this down by removing information that is no longer relevant to the core of your work.

**The gap**  
From the literature review and research plan. Why is your part of the problem different and worth solving? AND / OR What is different about your approach? What basis was there for using your approach?

**Method**  
From the research plan and notebooks. Note any methods that were abandoned and why. This section described what was actually done.

**Results**  
This is the result of any experiments you did using the tools you created.

**Discussion**  
How effective is your solution? How does it compare to other solution / does it solve the gap in the research (how effectively?).

**Future work**  
What further work can be done?

Keep these things in mind through the project and take notes useful to each section. (You may like to mark which section they might be useful to when you record them.)

**The development work**

Further information will be available from the SERE website on:

- RAISER 3: Versioning control
- RAISER 4: High level design of your project
- RAISER 5: Documentation of code
- RAISER 6: Reviewing code (and having your code reviewed)

**A note on RAISER**

RAiser is designed to assist you and reduce unneeded burdens during the research phase of your project (i.e. while you are working on it). To this end information for RAISER is release “as needed” and an effort is made to limit what you are required to do. The amount of contract and communication is also limited as much as possible.

If you feel you could do more, or would like more information or the planned information earlier please e-mail me: oboler@comp.lancs.ac.uk. While we don’t want you to forget about SERE and your use of RAISER… it should not be a priority, merely a tool available to you when you need it.

**12.2 Research Notebook**

**12.2.1 About the research notebook**

The research Notebook is recommended in the getting started guide and each year both these documents were either e-mail to participants or highly recommended to them via e-mail and in person. For an explanation of the Research note book, please see the guetting started guide.

**12.2.2 Template of the Research Note Book**

Research Notebook

<table>
<thead>
<tr>
<th>Date:</th>
<th>Name: __________________________</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>________________________________</td>
</tr>
<tr>
<td>Source:</td>
<td>______________________________</td>
</tr>
<tr>
<td>Notes:</td>
<td>______________________________</td>
</tr>
</tbody>
</table>

**e.g. 1**

Date: 22/6/03
Type: Discussion notes / ideas to do
Source: ICSERA conference question
X-reference: none
Notes: Question was on whether RAISER / RESET had been implemented and data collected in the results. This should probably be done!

**e.g. 2**

Date: 1/11/03
Type: Meeting notes / to do
Source: Meeting with Ian on research aims
X-reference: 22/6/03
Notes: Implement RIASER / RESET on student research projects. Need a control group and an experimental group in order to measure the effect RAISER and RESET implementation has.
12.3 Code Documentation Guidelines

12.3.1 About the documentation guidelines
The coding guidelines are in style based on a set of guidelines I took part in creating for the SORAL search and rescue project. In content they focus far more on ideas than on the function or style of code. They provide an easy way of integrating the capture of ideas with the coding process and keeping the two tightly coupled. The use of the dOxygen tool allows two very obvious benefits; a well formatted document based on the researchers comments, and a set of automatically generated diagrams that can be used in reports and presentations. Best of all, updates to the code can automatically be reflected in the documentation.

12.3.2 RAISER Coding Practise

Scope and Aims
This document outlines coding practice for use with the RAISER process. While some of these suggestions are general “good practice” other suggestions are specific to research coding or exist to enable RESET to occur more easily. There will be a separate guide for RESET coding practice.

This coding practise aims to make your code easier to adapt and to enable it to be used beyond the scope of your own project. It does this by encouraging the writing of readable, modular code with sufficient documentation for others to follow. At the same time the nature of research coding is recognised and every effort is made to minimize the amount of actual documentation work required, particularly while in the middle of coding new ideas.

Readability
Clarity and readability should generally take precedence over highly efficient (but hard to read) code. There are exceptions however these are rare and in any case would
only apply to a small part of the code. See “When efficiency is important” if you do have an exception.

**Layout and indentation**

While this differs with the programming language, the key is to keep things consistent.

Where possible:

- Use tabs instead of spaces to indent code
- Always line up the opening and closing parts of a segment of code

**Example 1:**

```c
ReturnType FunctionName(ParamType Param) {
    // CODE
} closers
```

- If using different machines for editing, ensure files are always left in the same operating system format (windows or unix format)
- Bracketing convention (if applicable) should be consistent.
- Code should conform to an 80 character width limit. This enables the code to be readable when printed (e.g. for review).

**Naming convention**

In short, have one.

A good convention is to name each word in the variable name with a capital letter (bar the first). E.g. aVeryLongVariableNameForALoopCounter.

Another important rule is to make variable names no longer than needed. The above variable could be renamed loopCounter for example. Names of less than 4 letters are typically not meaningful, with the notable exception of i, j and k, when used for loop counters. The variable loopCounter should therefore simply be called i. (The letter j and k should be used only when loops are nested.)

Where a loop counter is used for more than just checking the loops exit condition i.e. the value is retained and used later, use a more meaningful name than i, j, or k. This is to prevent the value being zeroed by accident in later coding.

**When efficiency is important**

When time *is* being invested in highly efficient code, a similar effort should be invested in documenting:

a) Why this method is needed (why is speed / disk space / memory conservation important at this point?)

b) How the method works.
   1. Internal comments should mark the start and end of the efficiency zone.
2. Preconditions (What is required to enter this section) and Post conditions (what is the guaranteed outcome e.g. what data transformation occurs?) should be listed.
3. External documentation should exist illustrating the method and how it works.
4. References used in creating the method should be listed (i.e. books, articles etc)
5. Where the source is a web page or a correspondence, a copy should always be retained with the project. In other cases it is recommended.

### Types of Comments

Comments in the RAISER process serve four purposes.

1. To keep track of the purpose of a module of code (What)
2. To keep track of the owner and version of a module (Who and When)
3. To keep track of the method being applied (How)
4. To keep track of the authors rationale (Why)

Comments occur at 4 levels,

1. File level
2. Class Level (if working in OO)
3. Method Level (OO) / Function Level (Structured)
4. Internal comments (these are inline comments)

Comments should preferably be compatible with dOxygen (download from the SERE website). The example comments later in this section are in dOxygen format. The dOxygen /todo comment (placed inside a comment block) should be used to indicate future work.

**Rationale:**
Like JavaDoc, this allows external documentation to be extracted from within the code files. The preference for dOxygen is because (a) It’s simple (b) It works for Java, C, C++ and other languages (c) the generated documentation comes in many useful formats.

### When to Comment

A good trick it to take a half day out every week / every few days (depending on your rate of progress) and use this to add any missing comments. Start with required comments, then go on to optional comments. After a few days (perhaps a weekend off) you should have a little distance between you and the code. Anything that isn’t immediately clear and obvious to you now need to be commented. If in doubt, ask yourself if your supervisor could understand this code without you there to explain it. Next thing of what questions they ask and document the answers, particularly the rationale ones.
**File Level Comments**

**What**
These comments name the project the file belongs to. They also given an overall description of what code in this file is about. They aim to quickly let someone know if (a) the code they want to write belongs in this file (b) the code they are looking for should be in this file.

**Who and When**
Where comments should be inserter into each file but should not need to be updated more than a couple of time. They record who is responsible for a major version of the file, and what version this is. The record will generally start with a “created” comment by the researcher. Typically the next entry would be “finalised” by the researcher. In the RESET phase other people will work on the code and add their names as needed to show that a particular take had been completed on this piece of code (e.g. “External Documentation RESET”, “Comments Reset”, “Refactoring Complete”, “Regression Testing Implemented” etc).

**Why**
It may make sense to comment on the overall structure of classes in this file and why they are structured like this. This can prevent the information being needlessly duplicated at class level.

**Class Level Comments**

**What**
These describe the specific responsibilities of the class. What is it for? This comment should be general enough to allow all things related to the class, and specific enough to distinguish it from any other similar classes. If you are unsure where a method belongs, this comment (in each class) should make it clear. The research should always add these.

**Why**
If a method appears not to fit anywhere this is a hint that it might be time to revisit the design, or at least add a why comment (at this class level) stating that method X doesn’t seem to fit the structure but has been included here because of reason Y. In some cases it may be appropriate to explain why a class exists (as opposed to being a collection of variable inside another class) or what the class is supposed to be abstracting.
Any alternative architectures for the class should be mentioned along with the decisions for and against them… future developments may make one of these a better option.
Method Level / Function Level
In most (but not all) cases during RAISER, a methods name will give sufficient information. In those cases where there is additional important information or where the exact result of the function is not obvious, the following suggestions apply.

What
What does this method / function do?
What constitutes valid/invalid input?
Are there any special cases?
What format is the output?
Is it scaled / rounded / ordered etc in anyway?

Who and When
If the author is NOT the author of the file, who is it? (e.g. supervisor, taken from a book, taken from a paper etc)
If this is an alternative implementation, a second try, etc note this along with where the original can be found (if the original worked and this is an improvement / alteration)

How
What algorithm is being applied?
What data structure is being used?
Are there any fudges? (if so what?)

Why
Relates to how. Why were these choices made? It is important to note if there is No Particular Reason (NPR). This means there may be better approaches that are open for future work. If you have any other ideas, but don’t have time or resources to try them, note these as well (making sure to note them as “alternative approaches - not tried”).

Internal comments
These should always be inline comments. This means that:
(a) The comments are generally only 1 line long and reside next to the effected line of code
(b) The comments are for use inside the code only and should not be extracted for external documentation.

Internal comments are always how comments.

How
These are the standard comments that describe how something is being done. E.g. How does this code work? What is this variable for? What does this result mean? Preconditions and Post-conditions are also useful. The amount of how comments should be sufficient for someone competent to make sense of the code and what it does. Do not neglect these comments. Do not over do them.
Sample Comments

These sample comments are in dOxygen format. Please use the format provided if possible as this will save time in the RESET phase. In any case, the types of comments listed above should be included as this will not only help you remember what your code does, but will also help you clarify your ideas and better fit your code to your architecture.

A full file header looks like this:

```c
/**************************************************************************
*   Global Project Name, Local Project Name
*   
***************************************************************************
/ ** \file filename.ext
 * \brief A brief 1 line description.
 * 
 * WHAT: An extended description of what code in this file is generally about.
 * WHY: This can include any general assumptions and rational about the approach used (the algorithm, sampling methods, search algorithm, heuristics etc) or the structure of things in this file e.g. breakdown into classes, functions, etc. It can avoid repeating the same assumptions etc at class level for many classes.
 * Each affected class should then reference this file header.
 * 
 * <b>Version History (Who and When)</b>
 * 
 * \verbatim
 * -----+----------+-----+--------------------------------------------
 * Who |   When   | Ver | What
 * -----+----------+-----+--------------------------------------------
 * Fred| 05/6/04  |  1  | Created.
 *     |          |     |  
 * \endverbatim
 */
```

A class header looks like this:

```c
(A break to make each new class stand out when scrolling through code)
/**** Customer Record Class ******************************************/
(A brief description of the class on a single line)
///  Responsible for customer details and staff discounts
(A more elaborate class description including the what and whys This description may span multiple lines.)
/**
 * WHAT: This class is responsible for all things related to
A customer’s record. It does not include staff information except in the form of a staff ID number to allow discounts, and an update function to synchronize with the personal records.

Why: Staff records are stored separately in the staffRecord class

```cpp
class customerRecord {
}
```

A method header looks like this:

```cpp
/** synchronize() ********************************/
/** A brief description of the method on a single line
/** // Removes staff IDs from customer records if staff no longer employed.
/** A more elaborate description if needed
/** *
/** The Staff ID of 1001 may be used repeatedly and is ignored by
/** this method. Use 1001 when lifetime staff discounts are given to
/** retiring long serving staff etc.
/** *
/** Why: This is a hack, but works fine. At some point is may be worth
/** adding a list function to show all those staff with ID 1001 to see
/** the system isn’t abused. Another check would be to include a
/** record of the staff member who created the record so any abuse can be
/** tracked.
/** */
void synchronize(void) {
}
```

An Internal Comment looks like:

```cpp
Result = init(2); // Result will now be 0 is vacant, 1 if free, 2 if on hold.
```

---

**dOxygen starter**

dOxygen comments use the format `/** comment */` and can run over many lines, or the format `///` that runs until the end of the line. These comments are treated slightly differently in addition to their difference in scope.

Bullets are created using `-` for unnumbered lists or `-#` for numbered lists. The `-` symbol must in either case be in the same column (character position) for it to be regarded as the same list level. (Alternatively, you can use HTML-style lists, and then have column freedom. HTML-style may be better for longer lists.)

Example:

```cpp
/**
 * A list:
```
* - Bullet Item 1
*  - # Numbered Item 1
*  - # Numbered Item 2 
*   - More info about Numbered Item 2
*  - # Numbered Item 3
*  - Bullet Item 2
*  - # Numbered Item 1
*  - # Numbered Item 2
*  
*  - Some other text.
*/

A list:

- Bullet Item 1
  1. Numbered Item 1
  2. Numbered Item 2
     - More info about Numbered Item 2
  3. Numbered Item 3

- Bullet Item 2
  1. Numbered Item 1
  2. Numbered Item 2

Some other text.

**Bold**

<**> This in bold </**>

**This in bold**

**Emphasis**

<**em**> This in emphasis </**em>

**This in emphasis**

**More at:** [http://www.stack.nl/~dimitri/doxygen/commands.html](http://www.stack.nl/~dimitri/doxygen/commands.html)
12.4 Summary Poster

http://www.comp.lancs.ac.uk/computing/users/oboler/sere/

Username:_____________________  Password ________________________

References:
Record relevant bibliographic details, add key notes to journal including page number where appropriate. Add to endnote, bibtex or other reference recording system.

Supervisor meetings:
Any major decision / major changes? What good ideas or suggestions did your supervisor make? Record them in your research notebook or in the supervisor meetings space in your process plan.

Commenting code:
What does the class/method do? Why are you doing it this way? What options have you rejected? Are you reusing anything? What suggest this approach (supervisor, paper, etc), how much effort have you putting into this bit of code? (i.e. first cut, or the n\textsuperscript{th} attempt to improve it). What else might you need to say about this functionality or bit of code for your report? Does this change your methodology at all, and if so have you updated your process model?

Process model:
Have you updated it this week? What’s changed? What can you further refine anything (or break it down to smaller steps)? Can you add target dates to anything? Have you linked your process model to examples or code comments that implement, explain or demonstrate your ideas? Have you reviewed the stages you have yet to do and looked for opportunities to start work for later on anything? Have you regenerated your dOxygen output (for code and process model)? Have you uploaded it to your website?

Evaluation:
Have you generated any results? Will anything you’ve done this week affect your results? Do you have enough time left for evaluation? Can you write-up / add to any results or methodology now?

Timing
Last day for intro to coding practise for research: Tuesday June 27th
Last day for review of process plan and initial review of coding: Wednesday July 5th
First day for final review of code/comments with a view to writing up: Monday July 31\textsuperscript{st}
Report deadline: September 8\textsuperscript{th}

CODING ONLY TIME: July 10\textsuperscript{th} to July 29\textsuperscript{th} – I’m not available except via e-mail, but really you should just be getting on with it during this period.
Questions? Please e-mail me: oboler@comp.lancs.ac.uk.
12.5 Re-engineering patterns Developed

12.5.1 Read Everything in Two Hours (Re-engineering Pattern)

“Read Everything in Two Hours” differs from “Read all the code in one hour” and from “Skim the Documentation in an Hour” by starting with auto-generated documentation and commented documentation, inserted in keeping with the RAISER Coding Guidelines. The documentation must be generated to ensure it is up-to-date. For the purposes of this research we have settled on dOxygen as our documentation tool of choice, this pattern is therefore based on this assumption for optimal benefit.

Intent
Make an initial evaluation of the condition and structure of a software system by reading dOxygen generated output and source code.

Example. You are facing a 200K line program written in Java to simulate a network of various types under various conditions. The code was developed as part of a PhD project in the area. The project used the RAISER coding guidelines. Other projects wish to build on the existing code. You have been asked to stabilise the code so it can be given to the various new researchers.

Context
You are starting a reverse engineering project of a large and unfamiliar software system developed for research by someone using the RAISER Coding Standard. The project is being stabilised so it can be used for further research projects.

Problem
You need an initial assessment of the internal state of the software to plan further re-engineering work and advise other research on the suitability of building on this code.

Solution
Working in a quiet room with out interruptions, generate the documentation. Spend an hour reviewing the online documentation and code, starting with the indexes for classes and variables to become familiar with the terminology and moving through the system’s class diagram to individual methods and source code as time allows. Finally review the files list and any explanations in there about structure.
After the hour’s reading, spend a further hour producing a report summarising your understanding of the project, the classes and how it all fits together. List any structural problems or potential trouble spots for readability, maintainability, adaptability, reuse etc. List priorities for what needs to be done to this code before it is ready for other researchers to use.

Hints. The time limitation should force you to extract only the most relevant aspects of the project. Problems should be either very important or very common. Below are some hints for the report:

- Large comments in classes probably indicate the driver class or conceptual level of the program
- Watch out for clarification on what variables are for, these may indicate a confusing variable name or a changed variable name. In either case it may be worth noting that comments should be all references to it in comments should be checked for correctness.
- Domain abstractions may be defined by classes high in the hierarchy. Child classes could indicate variations within the domain. Further variations may be possible – look for any comments indicating directions for future work and note these in your report.

Forces Resolved

Limited Resources. By applying this pattern you rapidly gain a familiarity with the code and with the documentation. This will allow you to work more efficiently later on.

Tools and Techniques. For optimal use this pattern requires that the code is in a language supported by dOxygen and the comments are in the proper format. If the idea behind the RASIER coding guidelines are followed however, the pattern should be useful even without the auto-generated documentation.

Reliable Info. By generating the documentation yourself you know the automatic parts are correct. The manual comments may be wrong, but the auto-generated parts should help you spot this discrepancy. Rationale comments give the only reliable source of information on what the author intended – anything else is guessing.
Abstraction. The generated documentation will include diagrams and listings at various resolutions. Comments may provide further rationale for the abstraction choices and explanation of what each class is supposed to handle.

Sceptic Colleagues. Some people will wish to start working on the code without any re-engineering. Though this pattern takes a small amount of time, the report can indicate quickly what sort of work needs to be done. Given that this method will be used by the RESET team, it is unlikely that any will complain about use of this method, thought hey may complain about the re-engineering work as a whole.

Rationale
Knowing the structure of the code as well as the rationale behind it is critical before any changes are made. The RAISER method provides the documentation, this patterns helps extract at least a summary of it right at the beginning of the re-engineering process.

Non-generated documentation may be out of date. It may also be accurate where written reports have been “fudged”. Unlike external documentation, when the documentation is stored with the code it is more likely to be kept up-to-date. Comments on rationale are also likely to have a longer lifetime than comments explaining a particular implementation. Any discrepancies should for the purposes of this pattern simply be noted for review at a later.
12.6 Installing dOxygen Guide

12.6.1 About the Installing dOxygen Guide
The guide was released to reduce the barrier to adoption of learning dOxygen and to get dOxygen set up in a way that is most useful for the generic research project. While the output can be tailored greatly, if students just want to get something set up quickly, this guide will allow that.

12.6.2 Installation of dOxygen
Download the dOxygen install file from the SERE website.

Download Graphvis install file from the SERE website
http://www.graphviz.org/Download_windows.php
Run both files.
12.6.3 Configure dOxygen
Run the doxywizard program from the start menu.

Use the wizard to get started. The first tab “Project” asks for basic details about your program. Make sure the “scan recursively” box is ticked.
The second tab “mode” requires some changes from the defaults. Select “all entities” and check the box for “cross-referenced source code”. Optimize your code for Java if you have coded in Java.

The third tab is “output” unless you are using latex for your report, untick it. When doing your report you may like to generate an RTF file (second bottom box) to convert your code and documentation to word format... then you can copy from their to your report rather than from the web version.
The last and most important tab relates to diagrams.

Change the setting so that you are using dot, and so all the boxes are ticked.

Click ok and you are done with the wizard.

Now click expert.
You have already configured most of these settings with help from the wizard. However, for images to work you need to both install dot, and tell doxygen where dot is. Click the right arrow until you get to the tab called dot.
This has already been set up for you with everything except the path to the dot program.

You may need to stretch the borders in order to get to some of the controls.

Click “folders” to get to the dot path.
The path you want is: C:/Program Files/ATT/Graphviz/bin (unless you installed graphviz in a different directory)

The DOTFILE_DIRS is the place dot files (to be included in your project) live. I’d suggest making this the same location that you store your doxygen generated files. I.e. The output directory you specified in the wizard.

Click ok.

SAVE your setup.
Again I’d recommend saving it in the “output directory” where you generated doxygen files will go. Your working directory is temporary space… this can be any where, but for simplicity the directory about your output directory is a good spot for it.

Click start and doxygen will run. To view the output, go to your “output directory” and open index.html

That’s it.

By changing the expert settings you can alter your configuration.. if you find any specific alterations useful and feel they would be good for others, please let me know.

e-mail: oboler@comp.lancs.ac.uk
12.7 Technical Review Preparation Guide

12.7.1 About the Technical Review Preparation Guide

The guide is meant as an aid to help students prepare for their technical review. There will eventually be 2 guides, one for a review of the process documentation and the other as a review of the process itself.

12.7.2 Preparing for your Technical Review

Your technical review is not something you are marked on. At the same time it is there to ensure you are proceeding with your work in a way that will be helpful to you later on. This means checking that you are recording vital information and doing other things like coding in a way that will make your life easier rather than harder when it comes time to write-up. It’s worth putting in a little effort now to save yourself a lot of effort later.

The technical review will take between 1 and 2 hours and will look at a number of different areas. Both you (the review subject) and me (the reviewer) must prepare for the review. For you this involved creating a few new documents and sending these along with your code and research journal to me via e-mail (preferably in a zip file with your name as the file name). For me this involves looking over your code, comments and other documentation and taking notes and making recommendations that we can then discuss.

The review can’t take place until you are ready for it, but as each review requires 2 to 3 hours of preparation they can’t all be done at the last minute. You are advised to book the earliest slot you are available for, and to ensure your material is e-mailed to me at least 3 days before the review.

Review dates:  3rd to 10th of August
              16th to 21st of August

Reviews after the 21st are unlikely to be productive for you.
E-mail your preference to oboler@comp.lancs.ac.uk
12.8 The Technical Reviews
The following plans for technical reviews were developed at the beginning of the SERE project. All except the first were deemed out of scope, though may be useful for future work. The first formed the basis for style of review used in SERE and explained previously in this thesis.

12.8.1 Technical Review 1: Improving the RAISER process
This review occurs while the research is in the early stage of coding their solution. The Producer is a researcher. The Reviewer is a member of the SDL.

Objective: examination, mainly looking for systemic faults in approach and inconsistencies with the guidelines.

Interaction Mode: asynchronous preparation (the researcher tried to match guidelines, the reviewer spotted deficiencies and creates corrected examples) followed by an in person meeting with access to a computer and any necessary software tools. The meeting will only involve the reviewer and the producer.

Technique: Checklist from coding standard followed by free review of the product with an eye to non functional requirements.

12.8.2 Technical Review 2: Introducing the Project to RESET
This review occurs when the research has finished and the decision that the research might have enough value to justify resetting it has been taken. The Producer is a member of the SDL. The reviewer is a senior member of the SDL.

Objective: exploratory, gaining familiarity with the code and its needs

Interaction Mode: asynchronous preparation (the researcher has now finished with the code, the reviewer goes over it according to the “Read Everything in an Hour” pattern [7]). A meeting with another engineering to discuss and present the report allows a second opinion and discussion on strategy.

Technique: guided review as described in the “Read everything in an hour” pattern. The review is a brief overview, not a detailed review. A recommendation to RESET or not (based on the impression of the code) must be given in the report and agreed post review.
12.8.3 Technical Review 3: Developing a re-engineered architecture

This review occurs once it has been decided to definitely RESET the product. The Producer is a member of the SDL who has produced options for re-engineering the product. The reviewers include a second engineer from the SDL as well as stakeholders in the RESET product.

**Objective:** examination, looking for areas that should be restructured at an architectural level.

**Interaction Mode:** asynchronous preparation which includes the production of new architectural designs and an explanation of rational for each of them in plain language. A meeting with another engineering (preferably the same one used in Technical Review 2) to discuss and review the alternative architectures. Anyone planning to build on the project is encouraged to participate in this as a reviewer, though the total number of reviewers should not be more than 5. (If there are more than 5, multiple reviews can be held).

**Technique:** Free review

12.8.4 Technical Review 4: Examining the RESET product

This review occurs once the product has been RESET. The Producer is the member of the SDL who RESET the product. The reviewers include stakeholders as describe in the interaction mode summary. This review may in fact be multiple reviews of various parts of the product with different members of the SDL being producers at different times.

**Objective:** examination of re-engineered code. In the Re-engineering Phase, we are concerned about the quality of the product but only insofar as this has an impact on the process of adapting it for new research work. This means that the quality of the product is not important in its own right, but only as a means of ensuring its scientific integrity, readability, comprehensibility, adaptability, reusability etc. In addition, in each of these non-functional requirements, the law of diminishing returns arises and a point is soon reached where the reengineering effort would be better applied to another project. When applying effort in this research it will be important to remember that this research is not concerned with the progress of a specific technology or
project, but rather with the progress of science as a whole or more locally with the entire set of projects.

**Interaction Mode:** asynchronous preparation by all reviewers. The code and documentation are to be reviewed. The review team should include: The producer, one other software engineer to review engineering quality, one researcher from within the field (but not involved with the project) to review scientific validity, a senior academic to review the cost / benefit situation, any researchers (or their supervisors) who intend using the code.

**Technique:** Free review within each person’s remit, with all reviewers looking for basic mistakes in documentation.

### 12.8.5 Technical Review 5: Consolidating the RESET phase

This review occurs once the RESET phase is over. The Producer is a member of the SDL responsible for overseeing the development. The reviewer is a senior academic responsible for signing off on the product.

**Objective:** consolidation

**Interaction Mode:** synchronous. The producer and senior academic review corrections made as a result of Technical Review 4 and with the aim of signing off the project.

**Technique:** Review only problems flagged in the last review and the RESET effort as a whole.

### 12.9 The Process Template

#### 12.9.1 About the process template

The generic template provided here is a java source code file. The code provides the structure and is not designed to actual “do anything” besides being parsed. The real work is in structuring the code and then in adding detail to the comments. This is the template that was used, and as can be seen, only two classes have been filled out.

#### 12.9.2 The Generic Template

```java
/*
 * GenericTemplate.java
 * *
 * Created on Jan 31, 2006
 * *
 * This template forms part of the SERE project. It contains all classes
```
needed to document the generic research process.

It may be adapted by participants to meet the needs of their project.

A living document and should be kept up to date.

To view the documentation you probably want to start at the top level. The ResearchProcess.

The documentation divides the research process into 7 key tasks. There is a usual order to these, but this is not shown in the diagrams (they are ordered in the code however).

Each key task is represented as a class. Within the class are attributes and methods.

The attributes general represent aspects of your approach, that is things you need to do. Sometimes these will include the detailed plan of what is required.

The methods generally represent generic tools you are using, or the details of a way you will do something. Where attributes are abstract ideas, methods are concrete plans.

Details ways of doing something.

This documentation is stored in a single Java file, GenericTemplate.Java. You can edit the file with any text editor, then regenerate the documentation. To learn how to use Doxygen and to generate the diagrams, see the RunningDoxygen
*/

/* NOTE: If you are viewing this code (GenericTemplate.java) in dOxygen the
comments that
* generate the dOxygen will be hidden. Only the structure will be visible.
*/

/// Top level of the research process, attributes are aspects of your work and methods
are tools and ways to get there.
/**
 * see \ref how_to_read_sec
 *
 * @author Andre Oboler
 * @version 1.0
 *
 * The presentation of the research process by class diagrams
 * both allows researchers to plan their process (selecting appropriate
 * tools early on in the process) and allows them to reflectively exam
 * their process, thereby improving it.
 */

public class ResearchProcess {

    // Generic top level processes. These can be further broken down into sub
    // process or include an explanation of how
    // to carry them out and what tools to use etc.

    /// How do I go about defining my research problem?
    private ProblemDefinition myDefinitionProcess;

    /// How do I manage my reading and record information for the literature review?
    Making a readme text file.
    private LiteratureReview myLiteratureReview;

    /// How do I create my research methodology? Seeing Andre. Using a template in
dOxygen.
    private MethodologyCreation myMethodologyCreation;

    /// How will I collect my result?
    private ResultsCollection myResultsCollection;

    /// How will I analysing the results?
    private AnalysisOfResult myAnalysisOfResult;

    /// How will I draw out conclusions and define my major findings?
private DrawConclusions myDrawConclusions;

// How will I create papers, my thesis and other products? What will I create?
private CreationOfOutput myCreationOfOutput;

// Generic top level actions. These are things that need doing throughout the
research process.
/**
 * Meeting with your supervisor is perhaps the most generic action.
 *
 * Supervisors Name:<br>
 * Room to meet at:
 */
public void meetWithSupervisor()
{
    /**
     * <h2>First meeting, date</h2>
     * 
     * <p>Notes on meeting...</p>
     * <p>These should be recorded in html.</p>
     */

    /**
     * <h2>Second meeting, date</h2>
     * 
     * <p>Notes on meeting...</p>
     * <p>Again these should be recorded in html.</p>
     */
}

/// How do I approach the task of defining the research problem?
/**
 * Describes the approach as well as the process descriptors (tools, guidelines, etc)
 * used to define the problem.
 */
public class ProblemDefinition
{
    /// How do I go about defining my research problem?
    /**
     * <h2>Keeping track of your research idea</h2>
     * The research problem will change over time, however there does need to be
     * a
     * stated problem to begin with. From here it changes and evolves as you learn
     * more and your focus shifts.
     */
    * My Initial problem statement is:<br>
}
* Date:
* 
* New problem Statement is:<br>
* Date:<br>
* Reason for change:
*/

int researchIdea;

/// Description given to you by your supervisor
/**
 * The supervisor may have given you an outline or a detailed description.
 */

public void supervisorProjectDescription()
{
    /**
     * <h2>Requirements</h2>
     * Note anything that is required. List it here so you can come back to it.
     */

    /** <h2>Suggestions</h2>
     * Note the suggestions, decide if any of these are of more interest to you
     * than any others. Add your own ideas and further developments to take back to your supervisor.
     * Add this information here!
     */

    /** <h2>Reading</h2>
     * Do the reading. Keep copies. Make sure all the bibliographic details are recorded.
     *
     * See LiteratureReview for more details.
     *
     * Keep notes on the reading here.
     */
}

/// The Getting Started Guide (from SERE website)
/** <h2>The Getting Started Guide</h2>
 * The Getting started guide is an introduction to the research process.<br>
 * It should be read early and followed from the start. <br>
 * It makes mention of the ResearchJournal and LiteratureReview<br>
 * It can be found here:
 http://www.comp.lancs.ac.uk/computing/users/oboler/sere/docs/starting.pdf
 */

public void gettingStartedGuide()
{
}
}
Dealing with literature and references, how do I do it?

* How do I find relevant background material?
* How do I record background information?
* How do I place my work within the context of past work?
* How do I add citations and create my bibliography?

The literature review task covers these tasks questions.

```
private class LiteratureReview
{

    // How to I find relevant background material?
    /**
    * An explanation of how you'll find material goes here.<br>
    * A good starting point is to ask your supervisor for something to read.
    * Once you've read it, look up some of the reference it refers to.
    * Once you've read them, look up what they refer to. Also using the web
    * search of papers that refer to papers you've already read.
    * Siteseer is good for this.
    */
    private int FindingMaterial;

    // How to I keep track of bibliographic data?
    /**
    * An explanation of how you'll Keep track of your references goes here.
    * The basic options are:
    * \li Paper cards - old fashioned but it works. Just don't drop the card index!
    * \li Basic softcopy. A \ref manualReference "list in a text file" or multiple
    * files divided by category.
    * \li Software such as \ref endNote (windows, MS Word) or \ref bibTex
    * \li embedded in a draft of the report / thesis or other output e.g. foot notes
    * I will keep track of bibliogrpahy data by... (you should fill this in when
    * you've decided
    * but be warned, putting the decision off will create more work for you!)
    */
    private int RecordingBibliographicData;

    // Manual reference list
    /** <h2>Manual Reference List</h2>
    * If you will will create a manual list, describe the process here in detail.
    * One option I have used in the past is creating folders for each main theme of
background research. Each softcopy paper I see gets saved into one of these folders.
Each folder also has a "source.txt" file. This lists all the bibliographic details related to that file. These files became redundant as references were added to EndNote, but are still a good way of looking over a theme and indexing the files.

What's your method? Replace this section with the approach you are using, if you have a manual method to supplement or replace the use of a tool.

```
public void manualReference()
{
}
```

// End Note (software - university license)
/** <h2>End Note</h2>
* This is a great tool if you have lots of reference. The university own a site license.
* Your supervisor or the department IT staff may be able to get you access to the software
* if it isn't already installed on the lab machines.
* End note is a database for references, and with one click you can add a citation to your word documents. It also formats them and creates a formatted bibliography.
* As you can't use both endNote and bibTex, one of these should be removed. To remove it
* perminently delete the method and the comments refering to it (i.e. this block including the short description (triple slash) above). To remporarily remove it, simply delete one slash from the tripiple slash and one star from the slash star star at the start of the comment. Then
* regenerate with dOxygen.

```
public void endNote()
{
}
```

// BibTex (software - Open Source)
/** <h2>BibTex</h2>
* This is another great tool and should be used if you are creating your document in Latex.
* If you are working in Linux you probably WANT to learn LaTex. But do this very early on
* It can take some getting used to.
*
* BibTeX allows you to create codes to refer to your references. When you compile your
document, Latex will change the codes to proper citations. It can also
generate a bibliography.

* For more details:
  * \li A quick guide
    http://cmtw.harvard.edu/Documentation/TeX/Bibtex/Example.html
  * \li More information http://www.csse.monash.edu.au/software/latex/
* As you can't use both endNote and bibTex, one of these should be removed. To remove it
  * permanently delete the method and the comments refering to it (i.e. this block including the
    * short description (triple slash) above). To remporarily remove it, simply
delete one slash from
  * the tripple slash and one star from the slash star star at the start of the
comment. Then
  * regenerate with dOxygen
   */
   public void bibTex()
   {
   }

   // How do I create my research methodology?
   /**
    * 
    */
   private class MethodologyCreation
   {
   }

   // How will I collect my result?
   /**
    * 
    */
   private class ResultsCollection
   {
   }

   // How will I analysing the results?
   /**
    * 
    */
   private class AnalysisOfResult
   {
   }
How will I draw out conclusions and define my major findings?

```csharp
private class DrawConclusions
{
}
```

How will I create papers, my thesis and other products? What will I create?

```csharp
private class CreationOfOutput
{
}
```

### 12.9.3 Output of the Research Process page

**ResearchProcess Class Reference**

Top level of the research process, attributes are aspects of your work and methods are tools and ways to get there. [More...](E:\research\Papers\drafts\cakes\talk\07\generic\class_research_process.html - r0)

#### Collaboration diagram for ResearchProcess:

![Collaboration diagram](legend)

[List of all members](E:\research\Papers\drafts\cakes\talk\07\generic\class_research_process.html - r0)

**Public Member Functions**

void `meetWithSupervisor` ()

**Private Attributes**

<table>
<thead>
<tr>
<th>ProblemDefinition</th>
<th>myDefinitionProcess</th>
</tr>
</thead>
</table>

How do I go about defining my research problem?

E:\research\Papers\drafts\cakes\talk\07\generic\class_research_process.html - r0
Detailed Description

Top level of the research process, attributes are aspects of your work and methods are tools and ways to get there.

see [How to understand this documentation at a glance](#)

**Author:**

Andre Oboler

**Version:**

1.0

The presentation of the research process by class diagrams both allows researchers to plan their process (selecting appropriate tools early on in the process) and allows them to reflectively examine their process, thereby improving it.

Definition at line 69 of file `GenericTemplate.java`.

**Member Function Documentation**

void ResearchProcess.meetWithSupervisor( )

Meeting with your supervisor is perhaps the most generic action.

Supervisors Name:
Room to meet at:

First meeting, date

Notes on meeting...

These should be recorded in html.

Second meeting, date

Notes on meeting...

Again these should be recorded in html.

Definition at line 104 of file GenericTemplate.java.

---

Member Data Documentation

**AnalysisOfResult** | `ResearchProcess.myAnalysisOfResult` [private]
---

How will I analysing the results?

Definition at line 88 of file GenericTemplate.java.

**CreationOfOutput** | `ResearchProcess.myCreationOfOutput` [private]
---

How will I create papers, my thesis and other products? What will I create?

Definition at line 94 of file GenericTemplate.java.

**ProblemDefinition** | `ResearchProcess.myDefinitionProcess` [private]
---

How do I go about defining my research problem?

Definition at line 76 of file GenericTemplate.java.

**DrawConclusions** | `ResearchProcess.myDrawConclusions` [private]
---

How will I draw out conclusions and define my major findings?

Definition at line 91 of file GenericTemplate.java.

**LiteratureReview** | `ResearchProcess.myLiteratureReview` [private]
---

How do I manage my reading and record information for the literature review? Making a readme text file.

Definition at line 79 of file GenericTemplate.java.

**MethodologyCreation** | `ResearchProcess.myMethodologyCreation` [private]
---


Definition at line 82 of file GenericTemplate.java.
ResultsCollection ResearchProcess.myResultsCollection [private]

How will I collect my result?

Definition at line 85 of file GenericTemplate.java.

The documentation for this class was generated from the following file:

- E:/research/sere/doxygen/process/GenericTemplate.java

### 12.9.4 My Personal Process Plan

While the full process plan is not needed to illustrate the point, Figure 19 provides a snapshot view of the process plan used in this work. Unlike the template, this more developed plan makes use of lower level diagrams. Future Work, Measuring Success, Components of SERE and Experimental Approach are all shown in red on the original and may be clicked. Figure 20 shows the lower level detail of the Components of SERE section. In some cases ideas and plans have been recorded, on others simply the structure and existence of a class provides sufficient information.

![Figure 19 Personal Process Model: The Research Process for this work](image-url)
12.10 The Case Studies

There seems to be some overlap as the effects looked at in “research case studies” will in some cases become costs in terms of case “Project case studies”. i.e. If in “Researcher Case Study” the effect is more time required when the treatment is applied, in the “Project case studies” this will be considered a cost in terms of developer time.

Differences:

“Research case studies” includes the subjective satisfaction factor of applying the treatment (compared to other projects in the context of this developer)

“Project case studies” include the objective quality with the treatment applied (compared to other projects with / without the treatment)

There results may be correlated or perhaps not be.

The individual cases will now be discussed. The later 3 are still being developed in more detail.

12.10.1 Researcher Case Study

Aim: Explore the effect on the researcher of applying RAISER (or not applying it)
Chapter 12  Appendix 2: Process Descriptors

Type: Multi case, holistic

BSH type: Blocked subject-project study

Unit of Analysis: A researcher

Hypothesis:

- RAISER is seen as beneficial by researcher
- Effort spent in RAISER is seen as being for the benefit of the researcher
- Effort expended in engineering is minimal compared to other projects
- Researcher encounters less problems or can recover from them better
- Improved focus on publishing

Selection of Pilot Projects:
Volunteers requested from the MSc class. Those who volunteer are divided into a control and experimental group. Both groups are provided information on how to do the software engineering required and use the recommended tools. The control group are left to themselves and not provided much motivation or external management. The Experimental group are managed so that they use the tools and methods provided, any recommendations are stated as mandatory for them. The division into groups is random, but sampling will be checked to ensure a mix of nationalities and academic abilities.

Description of method:
An introduction to RAISER / RESET for all students explains how research can be engineered, wasting the limited time available by doing conventional engineering. More often, students are told, research is instead not engineered at all. This introduction raises awareness of the problems. The introduction also explains the RAISER approach of using minimal engineering, but doing it in a way that gives immediate benefits and where most of the benefit is for the researcher themselves.

Volunteers fill in a short signup form including a survey on why they wish to participate. See (page 231). Further introductory material for participants (page 247) explains the importance of a research notebook (page 250). It also asks students to pick potential conferences and journals their work could appear in. The importance of a high level architectural design is stressed.
The cost of maintaining industry style software engineering documents is discussed in the introductory material, along with their limited use for a research project involving at most a couple of researchers. The exception students are told is high level architectural designs. In a limited number of meetings with both supervisor and researcher, additional engineering advice on the architecture and approach are given. This simulates the input a RESET laboratory might have into new projects.

Coding standards (pg 251) stressing the importance of commenting and a recommended coding style is given. The guidelines aim for more readable code. They also ask that code conforms to a standard suitable for use with the dOxygen tool and that rational is documented. Students are encouraged to comment where they have tried other data structures and algorithms and discarded them as this will act as a reminder for them in their report writing and more importantly prevent others wasting time on the same experiment out of curiosity. A lack of such rational may indicate an opportunity for future research or recoding for efficiency gains.

While still half finished the code is reviewed in a Formal Technical Review (FTR). FTRs are used to highlight weaknesses that may lead to a loss of knowledge, wasted effort, or other inefficiency by the current or future researchers. While defects will be detected and commented on, the primary goal is to improve the process of development. See “Technical Review 1: Improving the RAISER process” (page 271) for further details.

When the project is newly completed a survey is sent to all MSc students (see “Final Surveys” on page 234). A second survey is sent to participants (pg 236).

### 12.10.2 Re-setter Case Study

**Aim:** Explore the effect on the team who is resetting a project. What is the difference in effect when working on a project that used RAISER or a project that didn’t?

**Type:** Single case embedded design

**BSH type:** Multi-project study

**Unit of Analysis:** The RESET team
Embedded unit 1: A developer's work
Embedded unit 2: A role in the team, e.g. Engineer

Hypothesis:

- It is easier to RESET work that used RAISER
- The RESET process feels more complete when working on a RAISER product
- The RESET team can go to more depth when working on a RAISER product
- The RESET team feels that in a limited time a non-reset project can only be brought up to about the standard of a RAISER project (i.e. prior to RESET).
- The RESET team prefer to only work on RAISER projects.
- Embedded unit 1: Explore the effect of the various project and developer styles on the RESET team
- Embedded unit 2: Explore the impact of RAISER and of various styles in terms of individual team roles.

Selection of Pilot Projects:

Of those participants taking part in the project, at most 4 projects that have used RAISER and four projects that have not used RAISER will be selected for resetting. The selection will exclude any projects where the supervisors feels with work is either completed with no chance of a further project OR where the work done is deemed to be of such low quality that it will not be reused.

12.10.3 Project Case Study

Aim: To evaluate the result of applying RAISER / RESET to a product
Type: Multi case embedded design
BSH type: Blocked Multi-project study
Unit of Analysis: The product (x N)

Embedded unit 1: Application of RAISER
Embedded unit 2: Application of RESET

Hypothesis:

- The quality of the end product will be better in projects where RAISER / RESET is used
- The cost of applying RAISER will be examined
- The cost of applying RESET will be examined
• The costs of not applying it will be examined
  o Students will opt not to continued projects
  o Projects will reach a point where supervisors feel they can not be used further without a complete re-write
  o If worked on with neither RAISER nor RESET for a longer period of time the increase in effort to reset would be exponential
• Embedded 1: This will examine how RAISER improves quality (in the context of a project)
• Embedded 2: This will examine how RESET improves quality (in the context of a project)
• Students preference for projects will be measured, it is expected RESET projects will be preferred though with a limited number of RESET projects and only a small population choosing them this factor may be minor compared to the interests of the student and the possibility that no students interested in that area will be in the incoming group of students.
• Any publications resulting from MSc projects will act as a bonus to research quality, it is expected RAISER projects will have more of these
13 Appendix 3: Additional Results

13.1 Initial motivation
Before the experiment started students were asked if they wished to sign up. One of the questions on the initial sheet asked why the students wished to participate or not participate.

Students answers for participating in the first year were:
- “To learn more about how to use software engineering techniques to help me in my dissertation”
- “I was told it was going to make me work better both by my SE teacher and project supervisor”
- “I thought it will help me”
- “to make the most of the opportunity to properly engineer my project (?) [sic] and to learn more about S.E. process”
- “I think the experiment will help me structure my project work”
- “hopefully I will gain some useful skills”
- Interested to participate so that I can both help the experiment and possibly be helped by it”
- “I think it will be a good experience”
- “It appears particularly relevant to my project as it is in a research area. Some guidance would be appreciated.”
- “Felt pressured by department e-mails – there were loads of them. Felt guilty not to.”
- “To help out a fellow academic, plus was interested to see how RAISER / RESET would actually work.”

Students answers for not participating were:
- “Time”
- “No specific reason”
- “Not quite sure is it would take much time to learn SERE and do the experiment”
13.2 Preconception on Software Engineering
In the second cycle responses included: Two students included the idea of "Making software based on a software life cycle", another two included a similar idea of "A systematic way to develop software using engineering methods". Other responses include: "the techniques and practices involved in the design, construction, implementation and maintenance of software", with other students expressing similar ideas as “a systematic approach for designing, developing and deploying software based solutions” and “software engineering embodies the entire design and support process for the creation of software systems in a reliable and structured way - a reusable process.” Another answer was “the sequence of stages and procedures one must go through in order to create a useful and useable piece of software with the greatest rate of success”.

In the third cycle, other responses included "[the] title says it all - Engineering software, producing documentation and following processes", "the process used by programmers and project managers to take a software project from start to finish in a reliable, planned, efficient manner that satisfies both customer and developer", "A methodical and repeatable means of developing software systems to meet user requirements", “the management and process of managing the various stages of a piece of software's life cycle", "The process and relevant documents of producing software", "determines the flow how a software is made... so that it can work well in a real world scenario", "A methodology to build a more enhanced system architecture", "the discipline of undergoing projects which are intended to create and maintain a piece of software", "a sector of the computer science discipline which analyses the stages of a project and provides useful feedback", and "an IT discipline which involves socio-Technical and engineering techniques in order to lessen the software process's time making it work with more efficiency and less failures".

13.3 Usefulness of Software Engineering for Research
When asked about the benefits of software engineering for research, some students in both the second and third year were pessimistic. The third year provided more details which is presented here. A concern about changing requirements can be seen in two students responses: "Software made to explore a concept will often be forked in
several directions to explore different facets. It is **not possible to specify** all of these facets at the initial planning phase" and "You **don't know where you are heading**, so it is difficult to predict that and apply software engineering". The concern about restricting creativity was also expressed in two responses, "there should not be a **predetermined way of thinking** and implementing while working on research. Even a minor **restriction** might have a huge impact on the outcome" and that "Too much **overhead** when you really just need to think and make notes casually." One student felt it depended very much on the project itself, they replied, “hard to say. Could be useful in a project such as **system development**, but in some projects like **algorithm development**, don't think it is very useful”.

Optimistic views in the second year expressed benefits including:

- "Software engineering can help **managing time**, using and advancing existing software for your needs and providing **prototyping** and **evolutionary development** techniques. Though SE in research probably won't have the benefits that it would have in business, because teams are small, and often find new topics and jump off at a tangent and need to deviate from the 'model'."
- " it provides a **systematic approach""
- "A more 'formal' or engineering-oriented approach would definitely help and shape-up things. **Specifications and requirements** would come to light, users [would be] defined."  
- "In that way [using software engineering] we **discover more** about the problem as we design and specify the software"
- "The developer should follow the instructions and process in software engineering"
- "It **automates** some tasks which it was designed to replace and has benefits in cutting costs as well as being more efficient"
- "Software engineering can **bring structure""

In the third year benefits listed included:

Further comments included:

- "help to **structure** the task and attempt to break it down into steps”
- “encourage **intelligent thought** about the work”
• "Generic approaches in SWE can be applied in nearly every field, in particular interactive approaches; requirements engineering"

• "Some principles can be applied to enable **better planning** of **limited time** available. SE Techniques + knowledge provide a good basis for implementation"

• "It can help produce **modular, scalable software** that can later be **refined** based on commercial understanding"

• "It **paves the way for long term development** of the system and makes a **history of the system** which may be needed in the future"

• "Any piece of software with such high complexity will be likely to have a number of **complications** if SE techniques are not used properly. Even if SE is used there are **likely to be problems.**"

### 13.4 External factors

The following were the specific details of the external factors that affected two participants in the third cycle during their project.

The first student said, “external influences came in the form of a faulty fire alarm messing up my sleep patterns and hampering productivity, the freezer failing on no fewer than 6 occasions during the project, one of which coincided with ants rendering all my non-frozen food useless leaving me without food on a bank holiday weekend after [the shops] had shut.... - that didn’t help. No, I don't really expect that last paragraph was going to be useful, just wanted to rant.”

The other student also listed environmental factors as affecting their project experience, saying, “Noise: in the lab., in the residence  - negative impact. Air conditioning in the lab. during summer -- good.”

While relatively amusing, the effect should not be underestimated. In yet another similar case, broken cooking equipment was observed to cause a large amount of stress for one PhD students. The difficulty of obtaining food over the summer period is a general concern to postgraduate students as the campus partial shuts down once
the undergraduate students leave. These environmental factors should be addressed, but are outside of the scope of this thesis.

13.5 Additional final survey data

13.5.1 First cycle additional Data

This section provides additional data on the final surveys in the first cycle. The columns in these tables can be explained as:

- **Others code**: Was existing code used in creating the project?
- **SERE participant**: Was the student a participant in the experiment?
- **Extended**: Does the student feel their code can be extended?
- **Extended Comment**: Additional thoughts on someone extending the code
- **Easy to use**: Does the student feel their final code is easy to use?
- **Journal**: Did the student use a journal?
- **High level diagrams**: Did the student use a architecture or other high level diagram
- **Source Control**: Did the student use source control, e.g. CVS

Table 28 represents the data from participants in the experimental group, Table 29 from participants in the control group, and Table 30 presents data from non-participants.

<table>
<thead>
<tr>
<th>Others code</th>
<th>Extended</th>
<th>Extended Comment</th>
<th>Easy to use</th>
<th>Journal</th>
<th>high level design</th>
<th>Source control</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Extended - yes, see my report. Use for other projects - maybe, I can't think how directly.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>To an certain extent Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>as it was designed for demonstrating and evaluating existing concepts, I'm not sure how it could be extended for much else.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>YES-IT IS NOT COMPLETE YET BUT I AM GOING TO COMPLETE IT</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>It could possibly be extended but it is unlikely it could be used for other projects</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>think it could be</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 28 Final MSc Survey responses from the Experimental Group 2003-2004
Chapter 13  Appendix 3: Additional Results

### Table 29 Final MSc Survey responses from the Control Group 2003-2004

<table>
<thead>
<tr>
<th>Code</th>
<th>Design</th>
<th>Your Code</th>
<th>Non-participants</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Easy to use your code</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>I think so. Parts of the code are commented (although not a lot), but the structure follows to some extend example code from the libraries. Furthermore the core of my project is implemented as a library with defined interfaces.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes, no previous examples of applications written in C# which interface with Smart-Its.</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, but it was a prototype and does have bugs!</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, it could be used, but mainly as a reference guide when developing location aware applications.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes, but it was a prototype and does have bugs!</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

### Table 30 Final MSc Survey responses from the non-participant Group 2003-2004

<table>
<thead>
<tr>
<th>Others Code</th>
<th>SERE Participant</th>
<th>Extended</th>
<th>Source Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

#### 13.5.2 Third cycle additional data

The following provides additional data with detailed responses students entered on the final survey in the third cycle.

**Source Control**

When asked if they used source control, one student replied “No, a pen-drive, winRAR and regular backups did me just fine”. This shows that despite the answer being no, this student (a participant) was taking regular backups. Two other participants used IDEs that allowed limited revisions, one of these described the facility as “a limited session-independent history that occasionally works”. A non-
participant said they used “Darcs” and added “using CVS when not necessary (i.e. except when it’s the only VCS available on the open source project-hosting site you are using, or is already in use for a large project) is only for masochists”. Another student expressed similar views saying “recommending CVS ought to be a criminal offence”. These answers supports our decision to drop the recommendation to use CVS, but also demonstrate a number of other approaches to preventing data loss, and the fact that students (at least amongst the participant group) had taken the prevention of data loss seriously.

Journal use
One of the students (a non-participant), noted that they kept “a few notes, but mostly I relied on a good memory”. This makes the response only a qualified yes. Another student replied “sort of, in the form of my notes document which I have sent you, and in the form of my meetings record in the process document”. This hesitant response (also recorded as a “yes”) shows a very high level of information capture, as would normally occur in a journal, but using a the new SERE personal process model in place of a journal. Another student replied, “not formally, but one could be constructed from all of the task lists which identify what got done when... the information’s there, just in a different format”. In another ambivalent answer a student replied, “kind of, I jotted down notes at the end of every week to take to supervisor meetings but nothing overly detailed/formal”.

The conclusion drawn from this data is that the concept of a journal is sound, but the implementation was widely varied. The process model was a popular replacement amongst those who had access to it but other alternatives could also be investigated.

Use of software engineering
When asked in their final surveys if software engineering has a role in computer science research the students responded as indicated below.

Non-participants:
- “Yes because they can suffer from problems just as any other project can. Surely if using these methods can improve efficiency it would be a good idea”
• “I think for this size of projects students should use cutting-edge technology and should have access to tools that support software engineering in that environment”
• “I incline more toward the informal approach common in most open-source and free software (having written some myself, including some small-scale collaborative projects)”

Participants:
• “Yes it provides a discipline and a framework”
• “software engineering takes time and costs productivity... on large projects you may get paid back but this is far from given. By doing bits that give the most bang for buck in terms of time you can try to harvest some (most?) of the benefits without as great an investment in time”
• “Sure it does, but I feel that a) the 'standard' processes has to be modified, b) the small nature of many projects reduces the need for SE, unless they are to be reused in the future, c) SE is particularly necessary if we have larger projects, projects with multiple users, projects where quality is an important factor and projects that are intended to be reused in some form in the future. As for getting the highest marks in the MSc, I cannot say that SE provides a significant advantage, at least in general.”

Would non-participants sign up with hindsight?
Non-participants were asked if they would sign up with hindsight, the responses (all negative) were:
• “No, it sounds like something that would complicate work. I prefer to just do my own thing even if it might take longer - it's probably just a reluctance to use other people's tools”
• “No - I think I have an idea of how to approach projects of this size having done many of them before – I would listen to a 1-2hr presentation about it though, where I would hear the basic suggestions and pick whatever idea I like from it... I feel quite confident about my skills to succeed and produce quality work without though”
• “Not for this project. Perhaps for a different project.”
13.5.3 Final Participants Survey
The data for component utility in the first year was calculated as follows:

The various SERE process descriptors were rated on their usefulness by participants, the results are showing in Table 31. Only experimental group members could answer the last two parts of this section. The aim of these rating is to rank the process descriptors and determine which had the best overall utility. As already explain, the scale was 1=useless, 2=some use, 3=useful, 4=very useful, 5= the most useful bit. By examining the columns of Table 31 we can see immediately that Journal advice was the only column to receive a “most useful” rating, but it also attracted some “useless” and “some use” ratings. This pattern of mixed responses is repeated for all the process descriptors suggesting different tools and approaches were more suitable in different cases. This confirms the idea that there is no magic solution and a more tailored approach is needed. The idea of an ad-hoc approach that is not unplanned, but rather tailored to the project has already been discussed (pg 29), and will be come a theme of the students’ responses.

<table>
<thead>
<tr>
<th>Parts of SERE</th>
<th>getting started</th>
<th>journal advice</th>
<th>Tools website</th>
<th>cvs</th>
<th>coding guidelines</th>
<th>meeting with supervisor</th>
<th>FTR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>guide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 31 Usefulness of process descriptors 2003-2004

The two experimental group members who rated meetings and FTR as “useless” were not very involved in SERE (according to the data they provided). One stated “I did very little coding” while the other says they didn’t really benefit “but the onus was on me I guess”. For those who did put the effort in, the FTRs were consistently seen as “useful”.

To determine the utility we can average the score received for each part of SERE giving the results in Table 32.
Ignoring the two aspects (far right) only experienced by experimental group members, this ranks the remaining tools (those in the left hand box in Table 33) from most useful to least useful as: Journal advice, CVS, getting started, coding guidelines, and tools website.

To avoid bias as a result of individual preferences for scoring higher or lower, a second set of analysis was conducted that gave each participant a fixed and equal influence on the final results. This eliminates the differing personal scales of the participants by dividing each of their rating by their average rating. This is shown in Table 33. Note that the average for experimental group members is calculated overall their descriptors, including meetings and FTRs.

<table>
<thead>
<tr>
<th>getting started guide</th>
<th>Journal advice</th>
<th>Tools website</th>
<th>cvs</th>
<th>Coding guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.83</td>
<td>0.42</td>
<td>0.83</td>
<td>1.67</td>
<td>1.25</td>
</tr>
<tr>
<td>1.25</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>1.25</td>
</tr>
<tr>
<td>0.71</td>
<td>1.43</td>
<td>0.36</td>
<td>1.43</td>
<td>1.07</td>
</tr>
<tr>
<td>1.67</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>0.71</td>
<td>0.71</td>
<td>1.07</td>
<td>1.43</td>
<td>1.07</td>
</tr>
<tr>
<td>0.77</td>
<td>1.92</td>
<td>0.77</td>
<td>1.15</td>
<td>0.38</td>
</tr>
<tr>
<td>1.07</td>
<td>1.79</td>
<td>0.36</td>
<td>0.71</td>
<td>1.07</td>
</tr>
<tr>
<td>0.91</td>
<td>1.36</td>
<td>1.82</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>1.82</td>
<td>0.91</td>
<td>0.45</td>
<td>0.91</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 33 Weighted usefulness of process descriptors 2003-2004

Taking the average of these averages to compare the individual tools gives Table 34.

<table>
<thead>
<tr>
<th>getting started guide</th>
<th>Journal advice</th>
<th>Tools website</th>
<th>cvs</th>
<th>Coding guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.08</td>
<td>1.13</td>
<td>0.81</td>
<td>1.05</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table 34 Weighted average usefulness of process descriptors 2003-2004

Ranking the tools from most useful to least useful now gives: Journal advice, getting started, CVS, coding guidelines, tools website.

There is not much difference between CVS and the getting started guidelines and otherwise the order is preserved.
If we had included the final two items, Supervisor Meeting (0.98) and FTR (0.99), they would rank between the getting started guide and CVS. The values are shown in Table 35, along with their expression as a percentage and their rank. Note that the values provided for the final two components are based on dividing their scores by the same actor as for the other values (so this is no longer a true average). Had actual averages been the results would be Supervisor Meeting (0.95) and FTR (0.97), but the order would be unchanged.

<table>
<thead>
<tr>
<th>getting started guide</th>
<th>Journal advice</th>
<th>Tools website</th>
<th>cvs</th>
<th>Coding guidelines</th>
<th>Meeting supervisor</th>
<th>FTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>1.08</td>
<td>1.13</td>
<td>0.81</td>
<td>1.05</td>
<td>0.92</td>
<td>0.98</td>
</tr>
<tr>
<td>As % of weight</td>
<td>15</td>
<td>16</td>
<td>12</td>
<td>15</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>Rank</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 35 Weighted average of all process descriptors 2003-2004

The final order over utility, that is greatest benefit to the greatest number is therefore: Journal advice, getting started, FTR, Supervisor Meeting, CVS, coding guidelines, tools website.

13.5.4 Six questions in cycle three

This section provides a detailed analysis of students’ responses to the six questions mailed out to them as they were completing their projects in the third cycle. These results are summarised in the main thesis.

13.5.4.1 The Essential difficulties

The essential difficulties list is derived from the data below:

When asked about the problems they faced one participant commented that the most difficult aspect was “defining [aspect of task] and how to perform it. Basically the problem was that the model was too complex to sensibly write down on paper, even in very simplified cases, making it hard for me to understand what I was meant to do. That combined with a lot of hidden assumptions in the source material and issues with understanding difficult terminology made that a difficult week of 40-50 sheets of scribbles before coding anything. And I then had to abandon that code.”

The programming task itself presented challenges to student, as would be expected. A non-participant commented, “I think I've been fairly lucky in that my project hasn't
caused any major problems. Probably the most problematic aspect was producing [aspect], I had to learn a lot of new concepts and some fairly heavyweight maths which is not my strong point. Also I had to build a lot of the component in [language] of which I have limited experience.” A participant also found programming an issues saying their largest problem was, “[language] and my lack of use of it.” A non-participant commented, “handling… [project component] was vastly more complex than I'd anticipated”.

Another participant noted “Trying to [coding task], only to discover a limitation in [programming language,] a problem that doesn't seem to be documented. However, that was slack time at the end of the project rather than serious coding time. Obviously, the [background] research overrun wasted time, but was unavoidable due to there being a lot of research in the general area” This shows unavoidable time spent on both the coding and research side, but also highlights the fact that for this participant there was “slack time” at the end of the project. This same student in reply to a question on the difficulties they faced replied, “Too much research, not enough time... - Led to a bit of a rush at the end, although the process model and code commenting helped better plan coding/write-up in light of the limited time.” This comment about SERE process descriptors (the Coding Guidelines and Personal Process Model) added to other comments by this student (Case 0629) which said that SERE allowed them to exchange cheap time early on in the process for expensive time later on, shows a significant advantage of the SERE approach as well as its limitation on reducing only the non-essential tasks.

A non-participant commented, “I wouldn't say it was exactly wasted time but I did spend a large amount of time trying to get my head around [programming language components] which were quite tricky and as I didn't manage to achieve anything from it I guess that it was time that I could have used more efficiently.” This is an example of a research path that researched a dead end. There was no way of knowing this would happen until the work was completed, so it is only with hindsight that this appears a waste of time, and there is no way to prevent it.

One of the participant included an extensive list of things that “wasted time”:

- “The partial or exploratory implementation of features that went nowhere”
• “my supervisor's apparently contradictory instructions causing me to rewrite / restructure work to his changing needs”

• “my supervisor suggesting that I integrate with a tool of one of his PhD students; having attended a demo with the student I found that it was impossible as it does not meet the essential prerequisites”

• “Time before the project and at the start investigating different tools and approaches, such as bibliography management. OK I did use some and so it was a little fruitful, but there is still much time wasted than if I was just given the tools. But then I probably would've rejected them”.

This student also provided a section of specific comments on SERE, the concerns were very minor including a suggestion to add clip art to the cover that the student did not find amusing, comments that weren’t helpful 100% of the time, though the student notes “but this is not a total loss” as other comments were helpful, and a complaint about “trying to get me to fill in this form when in the middle of difficult final work on my project, though I rejected this [and handed the form in later]”. With the exception of the form (necessary at this point because some student leave the country / stop checking university e-mail very short after submission), these comments are all either essential or complaints that others could have done more of the students work for them. The self-realisation that investigating the options on tools personally was important is a key insight by the student into the need to manage their own process.

13.5.4.2 The Accidental difficulties

A non-participant commented that their main problem was, “a particular area of the coding which consistently failed and needed recoding”. A technical review might have helped resolve this problem once it was initially identified. In a set up with a RESET lab, a researcher would be able to consult both their research colleges, and the engineers to find a decent solution to this problem.

A participant who made little use of SERE commented, “due to [an] unclear project specification, all my work [undertaken] before starting [to code the] project [was] wasted and due [to] that I was in immense pressure before submitting [the] draft.” This student failed to make the jump from undergraduate style programming projects
to a research project. The student could have approached the project differently in a way that would not have wasted so much time at the start.

Over design can be a problem for research projects and is typically why researchers avoid industrial style software engineering. A non-participant noted they spent “Too much time spent on design which later on proved itself to be pretty pointless”. This shows exactly the sort of wasted effort SERE tries to minimize, in the case of design this is why we recommend only an architecture level of detail, the rest being too volatile in most research projects to justify the design time.

A combination of time pressures combined with a futile attempt at changing the environment can also lead to wasted time. A non-participant explained that in their case, “miscommunication regarding [resources] cost me over two days in the final moments of the project and trying to solve the issues myself was the biggest waste of time.” While better communication could have resolved the initial problem, once the problem occurred, this time could also have been better spent in other areas while others solved the resource problem.

Another problem is scoping. A non-participant noted, “I wasted a lot of time trying [approach]… [but] it didn't work out. Also, I spent time trying to make [one component] cope with things that obviously weren't going to be implemented in [another component] anyway.” This shows a problem of scoping and recognising which end point is being targeted.

Pressure towards the end was also a problem. One student commented on their reliance on outside assistance which let them down, this created stress in the final stages. The student, a non-participant explained, “my biggest problem was problems with the actual testing, which was delayed by a few days due to unexpected issues network issues, thus resulting in a seriously stressful final week.”

13.5.4.3 Assistance received and tools used
The non-participants did not appear to receive much help. Two replied they didn’t get any, one replied “Advice on the structuring of the report and feedback outlining
improvements” and the last replied “I got some advice on [research topic] and some PhD students helped me out with [technology for data collection].”

The participants by contrast all listed help received, two of them mentioning dOxygen, one said, “though, I was not able to schedule meeting at regular interval, I guess our first meeting where you introduced couple of tools like dOxygen were very useful.” The other simply replied, “dOxygen, dOxygen and dOxygen”. Another participant found reassurance most help, “by SERE: reassurance I was not completely off base (in technical reviews). Otherwise: my supervisor told me that my work was feature complete and I should only fix bugs and get on with the report.” The last found technical information of most use, commenting, “How to circumvent the problems I had with [programming language]. Also some advice on statistics.”

These responses show that participants did benefit from additional assistance, as would be expected. One non-participant mentioned visual studio and another mentioned Darcs as tools they used, other than dOxygen these was the only tool mentioned, with both participants and non-participants focusing more on the “advise received” part of the question.

Students were also explicitly questioned about their use of diagrams to model their system. Two of the non-participants replied that they did not use diagrams. One explained “not really, because of a lack of time to create one (and it would be so simple it would hardly be worth creating one anyways)”, this suggests this student felt such a diagram could only be done at the end, as time would not have been an issue earlier on. The other replied, “No. The software is… broken up into the usual obvious pieces [for this type of task]. [Due to the programming language used a] diagram isn't hugely helpful or important -- you can easily see how the modules interact by skimming the code (particularly since most modules explicitly state what's being exported).” Neither of these seem very good grounds for not having an architecture diagram. If the diagram would be easy to create or could be easily derived from the code, all the more reason for the research to quickly put such a diagram together and save others having to work the structure out from the code.
The non-participants who used diagrams replied, “I have lots of different diagrams of different parts of the system of varying degrees of detail. A detailed diagram of the whole system though would be quite difficult to produce” and “Yes, it's a component based system so the diagram just features all the main components, it doesn't include other things like framework classes or other objects in the implementation though”. Again both these responses show partial diagrams unlike before the reason given here seems to be that the code is too complicated.

All but one of the participants used at least a high level diagram. The participant who did not use a diagram had a largely non-coding project and replied, “No, not appropriate.” Those that replied yes showed a greater appreciation of the importance of a diagram than the non-participants who used a diagram, they commented: Yes but, “Not too detailed… It would’ve been nice to have dOxygen generate it for me, but I’m just being greedy. Of course, less detail increases the abstraction and makes the diagram more understandable”, “Yes: very simple 2 component diagram with 2 interfaces. But I do have 2 diagrams following these giving an overview of each component and the interaction of the 2 components.” Finally, “To be honest I worked very less as far as software engineering concerned. I had a very high level system architecture diagram and rest of the thing is just code!!!!” This last comment was from the student was less involved in SERE and under time pressures after expecting a full specification that was never given to them (this being a research project rather than the undergraduate or industry style projects they were used to).

13.5.4.4 The researchers’ process
Aspects of the process students were asked to comment on include:

- did you plan how you'd do things at the start
- did you plan before doing each step
- did you make lists
- did you discuss the project with people (what sort of people?)
- did you do more planning, coding or reading
- how useful was your initial schedule?
- what else had an impact on how you did your project?
- what sort of impact did it have (what did you do different, was this good or bad?)?
The following sections are rich in data and provide detail on the nature of the research environment for these students. There are many lessons that can be drawn from this data to help improve the research process, as well as observing how the process differs between researchers and between participants and non-participants.

**Non-participants**

Out of the four students, two were largely designed in code, one also largely designed before coding and one was designed incrementally as the project was developed. The following four passages reply the students experiences.

The first student replied, “I had a detailed initial schedule which was quite realistic because I managed to keep ahead of it this allowed me more time at the end to extend the system into further areas. I can't say I planned before each step, I'm the kind of person who likes to get all the planning and reading out of the way right at the beginning so once I get into the coding I know everything I need to. also once I've started building I like to keep the momentum going. I definitely did more coding, my project was very focussed on implementation. I think my project has stuck fairly well to what I planned from the beginning, there are no major differences in what I planned and the finished product.” This was a very straight forward experience with no dead-ends and no surprises. This may signify a more “low risk” or “less creative” problem which the research addresses.

The second replied, “I did a relatively detailed but high level design, this generally consisted of diagrams outlining the structure of components and how they interact. These designs were then shown to my supervisor where he would point out any ways of improvement. In accordance with the schedule I then spent approximately 8 weeks coding. After this was completed I began to write the report; after the competition of each section it was sent to my supervisor where he would make a number of modifications.” This project shows a high level of design work, and an attempt to iron out issues at the design rather than coding stage. It also shows significant supervision with both designs and thesis sections being reviewed.
Another student commented, “I had an idea what I'd have to do and how I would do it [at the start]. Yes [I planned each step], though the data collection suddenly exploded in size when I realized I had to add functionality to my tool.” The student made lists, “I had roughly 1-page lists of what I wanted to do at each step in the process (more like todo-lists, which defined high-level approaches and tasks, which I had laid out in more detail in my head).” The student discussed their project with a few non-technical people and received advice from a couple of programmers. “Coding and reading were the biggest portion of my work, because I had massive amounts of background reading to do and because of the nature of my project (very recent issue), quality publications are hard to find. Planning was less, because I had to learn [languages] from scratch, so I took a rather basic first shot at all the techniques I had to use eventually and then ended up rebuilding from scratch after three weeks of building, since by then I had mastered most of [the programming language].” The student commented that their initial schedule was “not as useful as expected - it did give a rough idea as to how much time is available, but essentially it was not detailed enough and couldn't have been, because it was impossible to estimate the learning curve and the issues of the [programming] environment.” The student also experienced “personal issues” which “had a rather negative effect, because some of this hit me very hard and caused me to lose precious time.” This student focuses on the coding and ironing out issues later than in the process. The need to rewrite the project may have been avoided, or the time reduced if more time was invested up front in learning the languages and designing the system, however if the first attempt was planned as such a learning experience, this may in fact have been the best way to proceed. Again we see both technical issues and human issues influencing the process.

“I ‘planned’ the earlier stages of implementation… but it was all totally obvious stuff. The more interesting work… was driven by a mixture of picking [a feature] and thinking about how best to implement it, and thinking… how to make their evaluation look more sensible. A very iterative approach. The initial schedule I had… proved roughly correct… but the refinement took longer than I thought. Discussion was fairly minimal, most people on the course don't seem to know enough about [this area]… Also, I mostly worked in my room. Mostly I coded. Some reading on related subjects… but not of it was really directly relevant.” Again we see the approach of development through coding. Reading on topics that turn out not to be relevant is, as
previously mentioned, part of the essential research task. The development again seems to be straightforward with features added as time allowed.

**Participants**

Within the four participants, the use of lists was more prevalent (three students commenting on it), two of the students felt they had done more reading than coding, one did not answer the question and the final student who was on an industrial placement felt they did more coding. The difference between the industrial project and the others is striking, the most obvious aspect being daily meetings with an industrial supervisor. The processes are presented in the remainder of this section.

The first participant had a very straightforward process. They explained, “I set about the project in the way described in the project proposal. Various hiccups on the way, but trawling through research literature was time consuming and frustrating at times.” Again an essential task that cannot be reduced, though students could be given a head start (moving the starting position) by giving them more articles in their area to start with.

A more detailed planning stage was used by the second participant. They explained, “I planned out most of the implementation decisions before even properly defining the project, - a couple of decisions got reversed after research, and a lot more decisions were identified and taken after that stage. Planning usually took the form of identifying/listing subtasks and deciding which I wanted done by when, basic but effective. More reading than coding.... planning is hard to define because with me being so interested in it I never really stopped.... my brain was always diverting at least some attention to it. The initial schedule was useful only in that it stopped my supervisor from complaining that I didn't have one.” The student also noted a number of external influences that made life difficult, these were relatively minor but cumulatively led to increased stress (appendix, pg 293). Environmental factors affecting student living conditions are beyond the scope of this thesis.

The third student explained how they started with a “broad outline”, they explained how this listed prerequisites and after these were met their approach was “code, code, code, then do evaluation and write-up.” Their approach during the programming
process depended on the problem they were solving. They explained, “difficult programming and report parts, I would plan out what I do on a separate sheet or inline. I would tend to reject these as I went on and understood the problem more intuitively. So it is a question of getting me started and showing how things will pan out, but they have little influence over the final result.” This shows the agile nature of the work and the need for approaches to like wise be agile. The use of lists and inline documentation is useful approach to this when compared to full upfront documentation. The student noted that once they set milestone dates they met their targets, but they also “also reject stuff that would cause me to exceed them, such as buggy feature work.” The student made extensive use of lists for bugs, ideas to solve problems, things to discuss with their supervisor, results of meetings, “todo” and “fixme” items, they also had a Research Process lists (the Personal Process Model) “where I put down ideas for things like what I am going to do for my evaluation.” Many of the other lists were observed to also be in the Personal Process Model or in the code as per the Coding guidelines. This student made extensive use of lists, updating them regularly, this was again observed in meetings and supports their answer. Asked if they discussed the project with anyone they noted that they discussed their process with the SERE Software Engineer, and technical details with their supervisor and a PhD student. Discussion with other students provided reassurance. The student had praise for their IDE, a “very good IDE allowing me to spend time on the logic rather than minor issues.” The student also provided along list of other factors ranging from features of their word processor to noise in the residence that had an impact (appendix, pg 293).

The final participant when asked if they planned things said, “I planned but it could not work out.” When asked about planning during the development process they replied, “Yes, I had discussion with my industrial supervisor on an average daily basis.” It should be noted that as this was an industry based project the requirements and environment were very different to that experienced by other students. The student confirmed that they made a “todo” list. Asked about others they discussed their project with, the student replied, “I was working away from others so not much but I had very good discussion with my industrial supervisor, which was very useful.” They felt they had done more coding than planning or research and that their initial schedule was very useful, both usual conditions and probably a result of the industrial
nature of the project. The student felt that the daily discussions with their supervisor had a large impact, they said, “it was extremely useful and avoided all the mistakes which I supposed [I would]… do.”

13.5.4.5 Making the research easier
Only three non-participants replied to this question. The feedback focussed on supervisor and resources. All four participants replied.

Asked what could make doing the research easier, only three of the non-participants replied. The first felt “some guidelines as to what the heck should be in a thesis/dissertation” would be useful. They also criticised the absence of their supervisor for three week, but noted it “wasn't a huge problem really, just irritating.” The second non-participant replied, “I can't think of anything, except for a subscription to expert-exchange” and the third echoed general satisfaction with the process saying, “honestly, can't think of anything. Everything went smoothly and the supervision was excellent.”

All four participants replied, the first said, “if initial project proposal was very specific, it saved my lots of time and due to that I get extra time to do explore other work.” The second felt the library catalogue system was cumbersome to work with, and the third joked “*cough* more time ;).” The final student also responded initially with humour, “joke answer: you doing it for us. Serious answer: provide guidance on all aspects, such as structure, what you should do and when you should do it, what you should do for styling, what you should include, what you should do for the demonstration, ... Best provided by the supervisor as otherwise it could be too generic. Perhaps the waste-of-time sheets they give out in the FYP for recording progress could be adapted here? But basically, a lot of these things are solved by hearsay and chit-chat amongst students. OK the SERE provides some of this, but by necessity it is too generic and not particularly comprehensive.”

13.6 The Case Studies
Table 36 presents an overview of the SERE components and which cases commented deliberately on them. Table 37 presents various benefits and links these to mentions of them in the cases. Table 38 presents a selection of other background factors relating to the cases.
<table>
<thead>
<tr>
<th>Case</th>
<th>Group</th>
<th>Personal Process Model</th>
<th>High level diagrams</th>
<th>Documentation &amp; dOxygen</th>
<th>Coding Guidelines</th>
<th>Technical Reviews</th>
<th>Version Control tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>0405</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0406</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0422</td>
<td>P (C)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0404</td>
<td>P (C)</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0417</td>
<td>P (E)</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0420</td>
<td>N</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0418</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0526</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0522</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0506</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0529</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0507</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0610</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0632</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0628</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0629</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0625</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 36: SERE Components

<table>
<thead>
<tr>
<th>Case</th>
<th>Group</th>
<th>Adoption</th>
<th>Cost Benefit</th>
<th>Knowledge retention</th>
<th>Systematic Approach</th>
<th>Increased Efficiency</th>
<th>Increased Reuse</th>
<th>Increased Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0405</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0406</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0422</td>
<td>P (C)</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0404</td>
<td>P (C)</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0417</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0420</td>
<td>N</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0418</td>
<td>P (E)</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0526</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0522</td>
<td>N</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0506</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0529</td>
<td>P</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0507</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0610</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0632</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0628</td>
<td>P</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>0629</td>
<td>P</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0625</td>
<td>N</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

Table 37: Cases and benefits

- indicated a negative mention (e.g. non-adoption). Cases and background
Table 38 Cases and project factors

- in mention of pressure indicated an extreme LACK of pressure towards the end
# reused their own code, but transferred architecture and language

13.7 Combined data without outliers

This appendix provides the data used is the results section for the combined second and third year results, but this time excludes the outliers, as was done in the second and third year individual year results.

<table>
<thead>
<tr>
<th></th>
<th>Mean Proj</th>
<th>STDEV Proj</th>
<th>Mean CW</th>
<th>STDEV CW</th>
<th>Mean P-T</th>
<th>STDEV P-T</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>64.17</td>
<td>8.22</td>
<td>61.01</td>
<td>6.46</td>
<td>3.16</td>
<td>6.35</td>
</tr>
<tr>
<td>P's</td>
<td>65.23</td>
<td>8.58</td>
<td>60.19</td>
<td>6.89</td>
<td>5.04</td>
<td>5.60</td>
</tr>
<tr>
<td>N's</td>
<td>63.19</td>
<td>7.92</td>
<td>61.77</td>
<td>6.07</td>
<td>1.42</td>
<td>6.61</td>
</tr>
<tr>
<td>P-N delta</td>
<td>2.04</td>
<td>0.67</td>
<td>-1.59</td>
<td>0.82</td>
<td>3.63</td>
<td>-1.01</td>
</tr>
</tbody>
</table>

Table 39 Summary Data for 04-06 without outliers

Table 39 should be compared to Table 18 on page 195. The medians are the same with the outliers excluded. The interquartile range of non-participant drops from 11 to 10.79.
Figure 21 should be compared to Figure 17 (pg 197). The difference is in the outliers, the participants still have a smaller spread than the non-participants below the box, however the non-participants now have a smaller spread above the box.

13.8 Approach to data collection

13.8.1 Interviews
Structured, semi-structured and free form interviews were all used. The structured interviews were used at the start and for post project interviews with non-participants. Structured interviews were chosen in these circumstances in order to gain background information from the interviewees and to explore specific themes across the sample. These related to between group case studies. Semi-structured interviews were used with participants during the experiment. Some sets of interviews had prepared questions tailored to the particular interview, in these cases the questions were based
on answers already provide by the student in surveys. The “Six Questions” in particular led to this sort of further examination of a topic. In some cases unstructured interviews occurred, often after another form of meeting (for example a technical review). These were used to solicit information on the process and general information on the impact of descriptors.

Interviews for this work were recorded on a digital recorder. An external microphone was not used, and the internal microphone was found to be sufficient for interviews of this type (with 2 or 3 people). The recorder was started when students entered the room and the students were informed that the session was being recorded. Interviewees were also asked to give their name, supervisor and project ‘for the tape’. This showed acknowledgement of the recorder without being overly formal. Attention was not drawn to the recorder again until the end. The use of a digital recorder that was integrated into a memory stick made this particularly easy as the recorder was unobtrusive and a typical item computer scientists would see in their environment. The recorder was not turned off until the student had left, and it would in fact be picked up and carried to the door as the student was walked out. At this point it was made clear that the recorder would not be turned off, yet many students still made valuable comments at the door that they had not made while the interview dynamics existed. This may be because they felt a throw away comment on leaving was less likely to lead to follow up questions or to make the interview go on for longer. In practice conversations at the door could go on for some time, and in some cases issues were picked up again at the next meeting.

Towards the end of the cycle selected interviews were loosely transcribed. The interviews were selected based on either the level of participation of the student (those who were more involved had richer data and were hence transcribed) or because it had been noted that interesting issues had been raised. This was consistent with the case study approach, as some interviews intersected more case studies and therefore had more value while others provided nothing more than confirmation. The ‘loose’ transcription refers to the fact that the initial transcription only recorded the theme being discussed at a given point in time, and an indication of where possibly useful quotes occurred. These were then filled in during a second pass. In the transcript
actual verbatim quotes were recorded in quotation marks and then checked back against the tape.

13.8.2 Surveys
Surveys were used to question the entire class at the beginning, the entire class at the end, and the participants at the end. The initial survey in the second and third cycle was done on paper in the introductory lecture. To prevent bias from future questions, the questions were included in the slides (one per slide) and not on the response form. The response form provided a question number and space to record the question and an answer.

The sign up form played the part of a very short survey as well, and students’ non-Lancaster e-mail addresses were collected at this early stage. This allowed students to be followed up (after they submitted) if they stopped checking their university e-mail address and had not submitted the final survey.

Surveys at the end were conducted via e-mail. One survey was for all MSc students, and included a section only for non-participants (to capture any indications of diffusion from the participants). The participants were also given a second survey (at the same time) which largely focussed on tool use.

The questions themselves are provided in Appendix 1, but in general closed questions were used to examine if certain phenomena occurred, followed by an open question soliciting further information. In general a five-point scale was used to determine impact (with the middle option being neutral). In some survey questions a three point scale (the middle being “the same”) was used. The three-point scale was used when magnitude was unimportant but direction of change (‘more to less’ or ‘less to more’) was being investigated.

The use of e-mail (with the questions sent as text) made replying very low effort, though in some cases led to student providing open responses rather than selected from the list of closed option presented to them. An online form could have avoided this, but may have seen a lower return rate.
13.8.3 Tags

The NVivo software (QSR International 2006) was used to tag and manipulate the data collected. Case nodes were created for each year of the experiment, and sub-case nodes for each student. These were named with the students name and case number. Elsewhere in the data only the case number was used.

Interviews were entered as “external sources” and a link was included to the audio file. This allowed the recordings to be played from within NVivo. The external sources were divided into folders by year, and then by interview set (e.g. initial in one folder, FTRs in another, and final in a third).

Tree nodes were used to store the tags. The top level tree nodes were: evidence collecting evidence, cost benefit analysis, subject’s research experience, the report (reference to the student’s dissertation), to include in thesis (items coded as significant quotations to be used), and tools. A number of sub codes were used under each heading. For example the “to include in thesis” had a sub-category for items used in the thesis. This helped track the use of data. Each of the SERE process descriptors were listed as a sub-category of tools. The subject’s research experience was divided into: problems, reuse, design and implementation, programming languages, documentation and a few other codes that had only minimal items coded against them. Cost benefit was divided into: preconceptions, benefit demonstrated, comments about “next time”, and comments on documentation (as a specific type of benefit demonstrated). The topic of evidence of collecting evidence was further divided, the sub-categories including a variety of types of evidence collected as a result of the technical reviews.

While the report by node gave a summary of information on each of these topics, more complicated queries were used as well. For example, one query extracted comments on documentation that were made by non-participants.

NVivo proved a valuable tool and without it or something similar managing the amount of data used in this project would have been impossible.
References

Alison, L. (2002). Interview with Dr Lloyd Alison. Interviews from the USE CSR project. A. Oboler. Melbourne, Monash University.
Charles Babbage Institute (2002). Who was Charles Babbage? Website of the Charles Babbage Institute, University of Minnesota.
Cook, S., H. Ji, et al. (2000). Software evolution and software evolvability, University of Reading, UK.


References


Meyer, B. (2002). Interview with Prof Bertrand Meyer, Interview conducted for the USE CSR project.


Oboler, A., S. Lock, et al. (2007). Targetted Improvements: focusing improvement to maximize benefits and minimize impact The First International Workshop on
References


Spolsky, J. (2002). The Law of Leaky Abstractions, Joel on Software


University of Oxford (2002). Computer Science: Oxford University Course Information.


