20 Years of Teaching and 7 Years of Research: Research When You Teach

Mike Holcombe, Christopher Thomson

University Of Sheffield
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK
{m.holcombe,c.thomson}@dcs.shef.ac.uk

Abstract. There are constant and increasing pressures on the time of all academics to provide top-class teaching. As many researchers have found this invariably leads to a drop in both research quality and quantity. At the University of Sheffield our pioneering observatory project shows how much needed empirical research can be combined with the teaching of software engineering. We present an overview of our research in this area which has investigated the effect of development methodologies, human factors, and more recently formal methods. We also briefly discuss our general methodology and point to further sources of information. It is our intention that this paper should provide a starting point for other researchers to use empirical techniques to validate their research in the own areas whilst teaching.

1 Introduction

There is growing pressure on academics to deliver both world class teaching and research across the board. In the UK this is as a result of: a more competitive recruitment environment; the introduction of student fees; and the research assessment exercise. As a result academics are spending more time on teaching and administration tasks whilst still trying to publish good quality papers on a regular basis.

At the University of Sheffield we have demonstrated that it is possible to combine subject teaching with the production of high quality research. This sleight of hand is possible by collecting empirical data to support our traditional research area. Empirical software engineering is the practice of collecting evidence that supports our claims for the technique or method devised. The good news is that with some creativity this research can be applied to most computer science subjects!

Our goal has been to devise experiments that can be used to assess the software engineering 'crunch' question: *does the proposed technique work in real projects?* We have achieved this through several innovative teaching courses which were devised twenty years ago. These courses focus on providing the students with a large scale project and real client, thus motivating the students and providing a valuable learning environment. It was as a result of changes to these courses seven years ago

that we realized that we could use them to see how methods were used on real projects.

The remainder of this paper: introduces the field of empirical software engineering; describes the courses and how they are taught; gives an overview of our research to date; and describes our research methodology.

2 Empirical Software Engineering

Empirical software engineering is the process of collecting evidence to show the value of some software engineering technique. The goal of this type of research is to amass suitable evidence that is convincing for the target audience of the research [1]. This may be to encourage take up of methods in industry, in which case the evidence should provide appropriate information for business leaders to allow an informed cost benefit analysis to be completed.

The empirical process is typically one of two types of experiment: observational or interventional. An observation study consists of a number of observations of software engineers using the technique under study. This type of study is typically used to discover or highlight things that may be of interest, such as Pfleeger and Hatton's case study which inconclusively showed that there was a marginal benefit to using formal methods [2]. Intervention studies compare two groups of developers that using contrasting techniques whilst trying to keep as many of the other variables under control as possible. This type of study is typically used to show that one technique is superior to another given the starting conditions, for example Sobel and Clarkson showed that students who had formal methods training and used those formal methods developed better software [3].

Empirical software engineering is still a growing discipline, with many of the standards not yet agreed or widely practiced. Several recent survey papers aggregate much of the knowledge to date [4-6]. In particular Zannier et al. indicate that whilst there have been improvements in the quantity of papers using empirical data, the quality of the analysis is still lacking [4]. Further more Hannay et al. indicate that there is little use of theory in explaining the outcomes of the research [5].

To be most generalizable empirical studies need to use subjects (the developers) from the population (for example professional developers in large companies) where you want the results to be valid. However it is not unusual to use students and student teams as subjects in empirical investigations [6]. The reasons for this are two fold, firstly some experiments have shown that in certain situations there is no significant difference between results obtained from students and those obtained from professionals [7, 8], and secondly the ready availability of students allows us to test a hypothesis at a low cost before approaching industrial partners.

The issue of cost is significant in any empirical study, as rewarding subjects for their participation can get costly, even when they are not actually employed [9], and may affect the results if not done carefully. This means that many of the studies in the literature are conducted over short time spans, and within very controlled circumstances. By applying such controls small sample sizes are more likely to give

significant results, which are easily interpreted. However these results can fail to take notice of real world pressures that exist in larger projects, so whilst a technique can be shown to be superior on paper it may not work in practice in combination with other effects [9].

Larger and less controlled projects model the real world more precisely but are prone to 'confounding factors'. These are the effects of uncontrolled variables in the experiment, which may either make results insignificant, or worse cause the interpretation of the hypothesis to be invalidated. Such larger studies are therefore most suited when a large number of subjects are present and the techniques which maybe used are constrained and possibility assessed. In many cases this is already done as part of the assessment of degree level courses.

3 Twenty Years of Teaching

A practical approach to the teaching of software engineering has been a feature of the degree courses at Sheffield for over 20 years [10-12]. Group development projects were introduced into the 1st and 2nd year in the late 80s. Formal methods and discrete mathematics have also been taught in the 1st year from the same period. The formal method chosen was Z and this has been a feature of the 1st year curriculum until very recently, supported by tools such as CADIZ [13, 14].

A key development early in this period was the move to involving 'live' clients in the 2nd year project class[12]. We introduced, each year, local business clients who then worked with the student teams on a key business problem and possible software solutions. Typically we would have each client working with 4-6 teams on their problem over a 12 week Semester (alongside their other courses). The teams would carry out business analysis on the client's business (we use the word business in a general sense, some of the client's organizations were charities, public sector organizations as well as commercial businesses) examine the 'business' objectives of the client and seek to identify requirements for a software solution, implement the solution and deliver it to the client. The client then evaluates all the solutions from the teams and chooses the 'best' which is then used in the client's organization. The winning team receives a small cash prize.

The discipline of building real solutions for real clients has transformed our knowledge of software engineering and the way we teach the subject. It has also enhanced our students experience to such an extent that employers are queuing up to recruit our graduates into very highly paid jobs.

This discipline, of building a real system, is a major shock to the beliefs and perceptions of academic software engineering teachers. Two very important issues confronted us immediately:

1. clients do not know what they want - requirements change quite dramatically during projects and this phenomenon lasts for several weeks
2. quality solutions are essential - we cannot deliver software that is full of bugs if it is to be used by the client in their business on a regular basis.

These critical issues focus the attention on two important processes:

−   requirements capture and the maintenance of the requirements document;
−   and quality assurance and testing.

Initially we encouraged students to use Z to formalize their requirements and UML to express their designs [15]. Unfortunately no team in our course that produced a complete and valid Z specification ever produced any working software. The UML designs were not maintained in the light of requirements 'creep' and provided little of value to the teams - investigations revealed that they produced these artifacts because we asked for them not because they helped in the development of a high quality solution.

Z has always been a topic that the majority of students disliked - they found it very hard to create high quality specifications for real software systems. Z has not been designed to make it easy to change as the requirements changes and it raises the question of whether this sort of model is ever relevant when the requirements are dynamic - as they are in most real software projects. Clients are unable to provide any feedback on a Z specification. A specification might be helpful in building test sets if it was up to date but test generation is not straightforward with Z [16, 17].

The UML suffers also from these problems - the diagrams are difficult to maintain, there is limited understandability with clients and little value for building tests [18].

These issues led us to develop a simple to use, lightweight formal method that produced specifications that could be changed during requirements elicitation easily; it could be understood easily by developers and clients and provided a basis for the generation of powerful system tests.

To do this we took some inspiration from hardware design where state machines have been a key component of the design process for many years and were test generation from state machine models is a well researched area. State machine are too simple for software systems and we have evolved notations and tools to support the use of more general machines - X-machines and later Extreme X-machines (XXM) [19-21]. These are now a key component of the agile methodology that we use in both the 2nd year Software Hut course and in the 4th year Genesys Solutions course - Genesys is a commercial software company run from the University's campus [22].

Initially we introduced X-machines in the 4th year and many students commented that this formal method was so much easier to use and much more valuable than Z so why did we still teach Z? This has resulted in the replacement of Z by X-machines in Year 1.

4 Seven years of research

There were never any formal experiments contrasting the use of Z or not - these would have been possible - but we were more concerned with delivering the client something.

We started to experiment when we introduced agile methods, following the Extreme Programming (XP) principles [23]. We introduced an agile approach because we did some maintenance in Genesys and found that the UML was completely useless

- it did not describe the final software at all since it had not been updated as the project went along.

Introducing XP was risky and we did it with half the class, the rest using traditional design-led approaches. Looking at the results from these experiments (Macias - who found that there was a slight, but measurable quality premium in using XP [24, 25]) we concluded that XP had some advantages - this was confirmed in further experiments (Abdulla - who discovered that teams that used XP well had a higher well-being [26-29]).

Since most of the projects done tend to be dynamic in the sense of the requirements changing, an agile approach - backed up by the lightweight formal method XXM have been appropriate and successful [22]. Some successful projects have been done using a more traditional approach but these have generally been where the requirements are stable - the most notable example is of a system that had been the subject of a contract with a commercial software house who did not deliver any software to the client - the requirements elicitation process, however, had been completed.

Having analyzed XP as a complete method we realized that there was a high degree of variance between the teams, often because they applied the approach differently. Therefore our continued experiments have examined individual issues in closer detail, but still within the wider projects. These observational studies have served to enhance of understanding of the variance and its place within a wider context.

Casual observations of the teams suggested that the quality of the delivered project was as dependant on the interactions between team members as it was on the method they followed and their skill. This led to detailed observations of the interplay of different personality types in development teams. Selected teams were observed in meetings throughout the project, to allow the creation of a detailed picture of team and individual behavior. Initial findings indicated that certain personality types do have a positive, negative or a combination of both effects on the well being of a software engineering team. In particular the results indicated that a team without sufficient discussion on pertinent issues runs the greatest risk of encountering serious project problems [30-36].

It is frequently observed in the popular press that changes to a software development project often lead to the failure of the software under development. Therefore a further study sought to gather evidence of theses changes within our own projects. This research found, unexpectedly, that the frequency of the introduction of changes followed a Rayleigh curve. Whilst this did not translate to the amount of developer effort spent in making changes, it offers an avenue for improved predictions [37].

Most recently we have been comparing the effects on product quality of testing early (at the time of coding) with testing later (shortly before delivery) in projects where the XP process was otherwise followed. In or student groups we found little difference between the two groups, in both cases an increase in time testing lead to higher quality software. However those using the test early approach invariably spent more time testing, thus delivering better software [38]!

5 Empirical Evaluations of Formal Methods

Extreme X-Machines (XXM) have become a feature of our software development courses allowing some empirical investigation of how they were used by the students, and also allowing for empirical evaluation of some of their properties. Two studies have been previously published of our experiments involving XXM [22, 37, 39].

Our first (informal) study investigated the adoption of XXM on our courses [22, 37]. In this study two forms of data were analyzed the XXM produced by the students and information collected from debriefing the students at the end of the projects. After first group used the XXM technique it was found that the students had trouble visualizing their systems as XXM. This was addressed by creating more examples, a more detailed tutorial, and a tool to help draw the diagrams. Further analysis showed that XXM could potentially work well in an XP development environment, which led us to investigate this more closely.

In our second study we proposed a metric which is easy to collect when XXM are used in the design process [37, 39]. This study utilized a set of diagrams which were produced with the tool; as such they were more uniform which allowed for a structured analysis technique. A formalized explorative observational cross-sectional study uncovered examples of the changes made to the XMM. Additionally the student teams were interviewed weekly to record the changes that had been made during client meetings. It showed that the frequency of XXM changes overtime followed a Norden/Rayleigh curve that was derived from recording the changed functionality proposed at the client meetings. Thus demonstrating the ability of XXM to capture real changes to the project and leading us to conclude that they provided an easy way to measure such change.

A current study is investigating client comprehension of various diagrams used by software developers to communicate their understanding of the system requirements. In this study two UML diagrams (use cases and activity diagrams) and an XXM diagram which represents a recently commissioned system will be shown to each client. The client will then be asked to evaluate the diagrams and provide feedback as to which diagram best describes their system.

6 Devising an Empirical Study in the Classroom

Experimenting in the classroom can be tricky if it is desired that the results of the experiment have strong validity in other contexts. We believe that these problems outweigh the benefits of being able to study projects in detail over a long period of time [40].

The first step is to identify a potential data set. All of our research has taken place in modules where the students have worked in teams over a long period of time (of at least 12 weeks at 15 hours per week). For the majority of the projects the students have also worked with external clients developing software that was actually put into use. Such realism is essential if we are to claim that our results are generalizable to a larger population. However more restricted projects could be used in order to provide initial results and fine tune the experimental method used.

Having found a data source an appropriate research question should be raised. This must take account of both the research and teaching objectives [7]. So if your proposed research area was to validate the use of a formal method, it would be best within a teaching module that was concerned with using the method. We have found that students can be particularly resistant to using a method if they see no clear use for it and also if they find its use is unsupported by appropriate educational materials. Indeed if the students resist the method to be studied strongly it may indicate that further development is needed. Some students are also dubious about methods and techniques that they do not see being used widely in industry – they look at the most common job adverts in the computing press and this influences many of their attitudes.

The next step is to define an experiment that will answer the research question. There are now several published guidelines for empirical research and reporting, Jedlitschka summarizes them and proposes an aggregated set of guidelines for controlled experiments [41], whereas Kitchenham has published a more general set [42]. However there is still continued debate on the best methods [43]. Intervention experiments are probably the hardest to achieve in the classroom setting when the experiment spans many weeks as it is difficult to control and record the different factors that affect each of the teams. However such experiments allow the effects of different techniques to be compared. In all cases if a statistical method is to be used it is wise to consult a statistician prior to the conducting experiment in order to validate the design and ensure enough data will be available.

Data collection is rarely straightforward, at a simplistic level we archive the team's working documents every week for later analysis, but this often conceals items of interest that are not recorded in this way [37]. Therefore the teaching staff also interview the students each week in the guise of a management meeting, these notes are recorded both for assessment and as a guide to explaining the collected documents where important information is missing. However it is not ideal to ask pure research questions in these meetings as these are more likely to introduced instances of the Hawthorn or Pygmalion effects [44-47]. These are where the people under study adjust their behavior to match the expectations of the experimenters. It is therefore preferable that the students should be focused on developing a project, as opposed to the research. We have aimed to overcome this by utilizing separate researchers (both PhD students and post-doctoral researchers) to carry out additional tasks such as distributing questionnaires, recording team meetings, and interviewing students on research specific topics. We know that we are doing this right when the students object to the interference of the researchers!

When the data has been collected it will need to be measured by some method. In the case of many standard surveys used there will be a defined way of calculating this, for your own you should consult your statistician. For documentation and interview notes a structured analysis technique such as grounded theory may also be helpful [48]. In the case of software artifacts there are a large number of defined metrics [49], for popular metrics and languages (for example: Java and the CK metric set [50, 51]) software tools have been defined to calculate these. However such metrics are often poorly suited to some research questions and so must be selected with care. In particular we have found the use of lines-of-code based metrics to be unhelpful whereas design based metrics have been more useful.

We are continuing to carry out research in alongside our teaching in the coming year we hope to be able to publish details of our experiments into: the comprehension of XXM diagrams by client; the effect of conflict on software engineering teams; and the relationship between motivation and team performance.

7 Our Method of Data Collection

At the University of Sheffield we have devised through experience a system of data collection that forms the basis for each of our experiments.

The main method of data collection is though a systematic collection of the teams' working directories. These are collected and archived on a weekly basis. This is in contrast to other research where this data has been collected through CVS [52, 53]. We found that the students used CVS in such away that summary data (as supplied by CVS) could not be trusted, we also found that there was a strong weekly cycle present where the CVS archive would be updated often by a single member on a regular day [37]. Informal interviews and student feedback also showed that they spent significant time learning CVS for little personal benefit. Therefore, as there were no research benefits to using CVS it was abandoned in favor of the weekly collection.

Having collected the files from the teams' directories various analyses were performed. The nature of the analysis was closely related to research questions understudy, however the most commonly calculated measurement was the lines of code produced this being calculated for performance over the whole development period and on a weekly basis [24, 37, 38]. In the case of the studies into XXM the changes made to the diagrams produced were identified and recorded on a weekly basis. In order to complete these tasks some simple command line tools were produced to automate repetitious activates, although much of the analysis was made by hand.

To support the measurements made on the code a tool was produced to record the development process that the teams followed. Whilst the teams were supposedly following well known methodologies it was clear that these were never followed closely [26]. The tool allows the developers to self document their process by recording minutes, tasks, diary entries, time sheets and story cards (following the XP methodology). The researchers can then search this data easily, with the additional benefit of being able to track changes to these documents.

Students however don't always run a project as expected, so it was necessary to collect further supporting evidence manually. Much of this process has been project specific to address particular research questions; however three methods of collection have been used successfully. Thomson and Huang both used short weekly interviews when collecting data to ensure that process information was not forgotten, in addition they also referred to the weekly summaries that the project managers (lectures) made after meeting the students [37, 38]. Much of Karn's research was focused around meeting observations where detailed notes were made of the subjects interactions [31]. Lastly both Karn and Syed-Abdullah used interviews and focus groups with individuals and teams at the end of the development projects to collect information [26, 31].

In order to place our work within the broader literature we have also found it necessary to use well known instruments. The most extensively used of these was Warr's well being measurement, with the Belbin, Myers Briggs and Big 5 personality tests also being used [54-57]. On occasion we have also devised our own questionnaires to collect specific data necessary to evaluate the research questions. However we have found that the students are reluctant to complete questionnaires, as a result we are cautious when evaluating these results.

Our final source of data is the marks that the students are awarded for their projects. In the case of the Software Hut these are especially useful as 50% of the marks are awarded by the client. The marks are used to derive two measurements: internal quality (based on marks awarded by the lecturers) that describes the quality of the code and process; and external quality (based on marks awarded by the client) that describes the quality of the product.

8 Experimental Setup

Our primary source of experimental data is derived from the Software Hut project. In this project the students are allowed to self-select their teams which are between 4 and 6 students in size. In the case of an experiment where treatments are compared the teams are allowed to choose the treatment numbers permitting. This allows us to split the class without any ethical concerns (about one group having an easier method than the other). In the case that the experimental design calls for observations or the completion of questionnaires we allow the students to opt out if required.

The development projects and clients are selected randomly from a list of potential clients who apply to work with the project. Typically we accept three clients for each presentation of the course, distributing the teams among them evenly, based on the preference of the teams. We place no restrictions on the teams on the technology or techniques that they use, other than those which are understudy. As a result we have projects completed in a variety of program languages with a selection of supporting materials.

 Where a longer study is required we typically use the Genesys project. By running over 37 weeks as opposed to 15 (including vacations) far larger projects can be tackled. In this environment the teams are selected to give a mix of skills and ability in each in each team. This ensures that the teams are more comparable; however each team completes a different project which each of which is of different complexity. Once again the students are free to select any tools and techniques which not part of the experiment.

9 Conclusions

Over last twenty years we have developed a set world leading courses where students on our degree program apply what they have learnt in an industrial like setting whilst retaining the support of the University. This program has made our students highly attractive to large employers who value the life-skills gained in these

settings. Severn years ago we observed that this environment was ideal for comparing the effects of different development techniques with a view to verifying claims made by their authors. In so doing we discovered that there were large differences in the ways that the teams applied the techniques, thus allowing us to appraise the value of these individual approaches. Our continuing investigations lead us to believe that human factors are a dominating feature of any development method.

We have successfully integrated our teaching practice with our on-going field of research. Within this unique environment we can study fairly large groups of developers over extended periods of time within a controlled environment. This is a useful resource as it has allowed us to study software engineering techniques in-depth within environments which similar to those found in industry. Whilst this research is no easy route, it has allowed us to verify claims made by others and suggest improvements to techniques with demonstrable evidence.

Such research is not without its problems and often the hardest problem is that of scale. To get statistically significant results a large sample is needed, which is often beyond the number of students on a degree program. The solution to this is to collaborate with other teams and universities, although tight collaboration is required for meaningful results [58].

Lastly we note that in the field of formal methods little work has been undertaken to validate the use of the methods on real world projects, as opposed to case studies. We feel that our experimental structure would be ideal for this form of verification, and may in time lead to results that maybe more convincing to industrial leaders. In this case a clear focus on showing the relationship between time spent on following the methods, dealing with changes to the specification over time, and the final quality of the product produced, will be required.

References

1. Segal, J.: The nature of evidence in empirical software engineering. In: Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on. pp.40-47, (2003)
2. Pfleeger, S.L., Hatton, L.: Investigating the influence of formal methods. Computer, vol. 30, pp. 33-43. (1997)
3. Sobel, A.E.K., Clarkson, M.R.: Formal methods application: an empirical tale of software development. Software Engineering, IEEE Transactions on, vol. 28, pp. 308-320. (2002)
4. Zannier, C., Melnik, G., Maurer, F.: On the success of empirical studies in the international conference on software engineering. In: ICSE '06: Proceeding of the 28th international conference on Software engineering. pp.341-350, ACM, (2006)

5. Hannay, J.E., Sjoberg, D.I.K., Dyba, T.: A Systematic Review of Theory Use in Software Engineering Experiments. Software Engineering, IEEE Transactions on, vol. 33, pp. 87-107. (2007)

6. Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N., Rekdal, A.C.: A Survey of Controlled Experiments in Software Engineering. Software Engineering, IEEE Transactions on, vol. 31, pp. 733-753. (2005)

7. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in using students in empirical studies in software engineering education. In: Software Metrics Symposium, 2003. Proceedings. Ninth International. pp.239-249, (2003)

8. Host, M., Regnell, B., rn, Wohlin, C.: Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empirical Softw. Engg., vol. 5, pp. 201-214. (2000)

9. Sjøberg, D.I.K., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., Koren, E.F., Vokac, M.: Conducting realistic experiments in software engineering. In: Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n. pp.17-26, (2002)

10. Holcombe, M., Stratton, A., Fincher, S., Griffiths, G.: Projects in the computing curriculum. In: the Project98 workshop. Springer, (1998)

11. Kalra, B., Thomson, C., Holcombe, M.: The Software Hut "A Student Experience of eXtreme Programming with Real Commercial Clients". In: Extreme Programming and Agile Processes in Software Engineering, LNCS, Vol. 3556. Springer, pp. 323-324 (2005)

12. Holcombe, M., Parker, H.: Keeping our Clients Happy: Myths and Management Issues in 'Client-led' Student Software projects. Computer Science Education, vol. 9, pp. 230-241. (1999)

13. Potter, B., Sinclair, J., Till, D.: Introduction to Formal Specification and Z (2nd Edition). Prentice Hall PTR (1996)

14. Toyn, I., McDermid, J.: CADiZ: An Architecture for Z Tools and its Implementation. Software - Practice and Experience, vol. 25, pp. 305-330. (1995)

15. Unified Modeling Language (UML), Object Management Group, http://www.omg.org/technology/documents/formal/uml.htm.

16. Burton, S.: Automated Testing From Z Specifications. pp. (2000)

17. Laycock, G.: The Theory and Practice of Specification Based Software Testing. Thesis, University of Sheffield (1993)

18. Simmonds, J., Van Der Straeten, R., Jonckers, V., Mens, T.: Maintaining consistency between UML models using description logic. In: RSTI srie L'Objet, Langages et Modles Objets, LMO'04. pp.231-244, (2004)

19. Eilenberg, S.: Automata, Languages and Machines. Academic (1974)

20. Holcombe, M., Ipate, F.: Correct Systems - building a business process solution. Springer-Verlag (1998)

21. Thomson, C., Holcombe, W.: Applying XP Ideas Formally: The Story Card and Extreme X-Machines. In: 1st South-East European Workshop on Formal Methods. pp.57-71, South-East, (2003)

22. Thomson, C., Holcombe, M.: Using a formal method to model software design in XP projects. Annals of Mathematics, Computing and Teleinformatics, vol. 1, pp. (2006)

23. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional (2004)

24. Macias, F.: Empirical Assessment of Extreme Programming. Thesis, University of Sheffield (2004)

25. Macias, F., Holcombe, M., Gheorghe, M.: A formal experiment comparing extreme programming with traditional software construction. In: Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on. pp.73-80, (2003)

26. Syed-Abdullah, S.: Empirical Study on Extreme Programming. Thesis, University of Sheffield (2005)
27. Syed-Abdullah, S., Holcombe, M., Gheorghe, M.: The Impact of an Agile Methodology on the Well Being of Development Teams. Empirical Software Engineering, vol. 11, pp. 143-167. (2006)
28. Syed-Abdullah, S., Holcombe, M., Gheorghe, M.: Empirical Study on An Agile Methodology: An Action Research Approach. In: the 8th Annual Conference on UK Academy for Information Systems, PhD Consortium. (2003)
29. Syed-Abdullah, S., Holcombe, M., Gheorghe, M.: Practice Makes Perfect. In: Lecture Notes in Computer Science. pp.354-356, LNCS, (2003)
30. Karn, J., Cowling, T., Syed-Abdullah, S., Holcombe, M.: Adjusting to XP: Observational Studies of Inexperienced Developers. Extreme Programming and Agile Processes in Software Engineering, LNCS, Vol. 3556 Springer, pp. 222-225 (2005)
31. Karn, J.S.: Empirical Software Engineering: Developer Behavior and Preferences. Thesis, University of Sheffield (2006)
32. Karn, J.S., Cowling, A.J.: An Initial Observational Study of the Effects of Personality Type on Software Engineering Teams. In: the 8th International Conference on Empirical Assessment in Software Engineering. pp.155-165, (2004)
33. Karn, J.S., Cowling, A.J.: Using Ethnographic Methods to Carry Out Human Factors Research in Software Engineering. In: Measuring Behavior. (2005)
34. Karn, J.S., Cowling, A.J.: A Study into the Effect of Disruptions on the Performance of Software Engineering Teams. In: the 4th International Symposium on Empirical Software Engineering. pp.417-427, (2005)
35. Karn, J.S., Cowling, A.J.: A Follow Up Study of the Effect of Disruptions on the Performance of Software Engineering Teams. In: the 5th International Symposium on Empirical Software Engineering. (2006)
36. Karn, J.S., Cowling, A.J., Holcombe, M., Syed-Abdullah, S., Gheorghe, M.: The Positive Effect of the XP Methodology. Lecture Notes in Computer Science, vol. 3556, pp. 218-222. (2005)
37. Thomson, C.: Defining and Describing Change Events in Software Development Projects. Thesis, University of Sheffield (2007)
38. Huang, L.: Analysis and Quantification of Test First programming. Thesis, University of Sheffield (2007)
39. Thomson, C., Holcombe, M.: A Design Change Metric Derived From Extreme X-Machines In: 1st South-East European Workshop on Formal Methods. SERC, Thessaloniki, Greece (2007)
40. Holcombe, M., Cowling, A., MacÃ-as, F.: Towards an Agile Approach to Empirical Software Engineering. the ESEIW 2003 Workshop on Empirical Studies in Software Engineering, vol. pp. 37-48. (2003)
41. Jedlitschka, A., Pfahl, D.: Reporting guidelines for controlled experiments in software engineering. In: Empirical Software Engineering, 2005. 2005 International Symposium on. pp.10 pp., (2005)
42. Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng., vol. 28, pp. 721-734. (2002)
43. Kitchenham, B., Al-Khilidar, H., Babar, M.A., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., He, Z., Liming, Z.: Evaluating guidelines for empirical software engineering studies. In: Proc. 5th ACM IEEE Int. Symp. Empiricl Softw. Eng. pp.38-47, (2006)
44. Gillespie, R.: Manufacturing knowledge: a history of the Hawthorne experiments. Cambridge (1991)
45. Mayo, E.: The human problems of an industrial civilization. MacMillan (1993)
46. Roethlisberger, F.J., Dickson, W.J.: Management and the Worker. Harvard (1939)

47. Rosenthal, R., Jacobson, L.: Pygmalion in the classroom. The Urban Review, vol. 3, pp. 16-20. (1968)
48. Strauss, A., Corbin, J.: Basics of Qualitative Research: Techniques and procedures for developing grounded theory. Sage (1998)
49. Fenton, N., Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. Thomson (1997)
50. Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, vol. 20, pp. 476-493. (1994)
51. ckjm - http://www.spinellis.gr/sw/ckjm/.
52. German, D.: Mining CVS repositories, the SoftChange experience. In: International Workshop on Mining Repositories. Edinburgh, Scotland, UK (2004)
53. Liu, Y., Stroulia, E., Wong, K., German, D.: Using CVS historical information to understand how students develop software. In: International Workshop on Mining Repositories. Edinburgh, Scotland, UK (2004)
54. Belbin, R.M.: Management Teams: Why they succeed or fail. Butterworth-Heinemann, Oxford, UK (1981)
55. Costa, P., McCrae, R.: Four ways, five factors are basic. Personality and Individual Differences, vol. 13, pp. 653-665. (1992)
56. Myers, I.B., Myers, P.B.: Gift's Differing: Understanding Personality Type. Davis Black Publishing, California (1987)
57. Warr, P.B.: The measurement of well-being and other aspects of mental health. Journal of Occupational and Organizational Psychology, vol. 63, pp. 193-201. (1990)
58. Jedlitschka, A., Ciolkowski, M.: Towards evidence in software engineering. In: Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on. pp.261-270, (2004)