Research Documentation Guidelines

Capturing knowledge, improving research

Andre Oboler
Computing Department
Lancaster University
Lancaster, UK
oboler@comp.lancs.ac.uk

Ian Sommerville
School of Computer Science
St Andrews University
St Andrews, UK
ifs@dcs.st-and.ac.uk

Abstract

This paper introduced coding guidelines for use by academics developing code as part of their research in areas of computer science or similar disciplines. We introduce the guidelines and discuss their success and popularity as a tool for MSc students undertaking five month research projects. The guidelines lead to the use of comments combined with dOxygen as an agile approach to model both the software and the research ideas as they develop and change.

I. Introduction

This paper presents our Research Documentation Guidelines for use by researchers in university computer science departments. The guidelines aim to capturing the extra ideas and information that would otherwise be lost when a research project comes to a close. Our implementation combines the guidelines with dOxygen, a JavaDoc like documentation tool for Java, C++ and other languages and tests the adoption and results using multiple five month MSc Project over the course of three years.

The documentation guidelines are one tool in a RAISER [1] development process that aims to improve productivity for the current researcher as well as improving the quality of the software and data collected to assist future researchers. The RAISER development process is an SDLC specific to Software Engineering by Computer Science Researchers [2] to meet the needs of their research environment. The RAISER/RESET approach splits the long-term work into Research (carried out by researchers under a RAISER methodology) and Development (to be carried out by professionals engineers attached to an academic institution under RESET guidelines). The coding guidelines were tested in a way that simulated the availability of a software engineering with experience in RESET, though no RESET work was conducted.

In our work we aim to develop approaches that meet the RAISER guidelines and experimentally test them. The Documentation Guidelines are one of our oldest tools and have been used over a three year period with increasing success. Success for our approach can be measured along two axes, the perceived benefit by researchers and the adoption rate. The null hypothesis states that the default unplanned approach (without the aid of the documentation guidelines or other similar tools) is equally good and the only approach researchers find acceptable i.e. researchers see no benefit in the approach and it is either not adopted at all or found to be a burden with higher cost than value. We aim to disprove this hypothesis.

Our work uses MSc students engaged in research projects at Lancaster University. As very early stage researchers, MSc students were seen as more likely to try new approaches. As students with hard deadlines and project that only last about 5 months they were also seen as being very discriminating when it came to their own cost / benefit analysis of potential tools. Successful adoption of a tool is itself a validation of a tool having greater benefit than cost. Our case studies also involved surveys, interviews, observation and analysis of students' final products. The use of Documentation Guidelines, student perception of their usefulness and the changes to practices and product that the caused were monitored throughout the experiment.

We begin this paper with a discussion of the research environment, followed by an introduction to our experimental basis. Next we examine the problems we hope to solve through the use of documentation guidelines and the rational for using the guidelines we've experimented with as the solution. The guidelines themselves are then introduced followed by a discussion of their adoption and the both the researcher and engineers view of their success. We end with a discussion of future work including the possibility of a software tool to augment our approach and our conclusions on the guidelines as a response the problems introduced.

II. The Research Environment

The definition of research used by the Organisation for Economic Co-operation and Development (the OECD) is "creative work undertaken on a systematic basis in order to increase the stock of knowledge" [3]. The tension is between allowing researchers the freedom to creatively explore and ensuring there is a systematic approach. Meeting both goals is a challenge, and one that is usually left to the researcher. Having examined the research environment we agree that care should be taken not to step on academic freedom, yet the development of software, even for researcher purposes, remains a problem that can be helped by sound engineering. Applied properly, Software Engineering can provide the systematic basis that separates real research from hobby coding. As industry looks more to extreme programming and lighter types of engineering, it becomes possible to again consider the ideas of software engineering that grow out of industry – or at least the concepts we teach our students in order to prepare them for industry – and consider how they could be applied to our own research based problems.

Speaking about software engineering in general, Glass [4] suggests improvement will come from 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" [5]. The lack of planning is discipline specific and not part of the general usage of the term. The Latin root of ad hoc means "to this", an approach can be planned (in advance) yet still be a tailored solution.

The coding standard we introduce here is part of a wider set of tools aimed at researchers working in a university environment on small (one to two people) projects. We aim to allow personalised software engineering that is still systematic. We believe this combination best meets both the needs of the computer science research environment and the definitions of research commonly used by funding bodies.

III. Experimental basis

Between 2003 and 2006 we provided an opportunity for MSc students to participate in an experiment offering additional tools and methods to assist them with their five month research projects. In an evolutionary manner, the tools and methods were updated before each group started work. In the final two years 21 students opted in (our participants), and 28 students opted out. One of earliest and most stable artifacts were guidelines on documenting research code.

We gathered results on participants through observation of students at work during the project, inspection of final products, formal technical reviews of code and comments, and a post project survey on the tools. Additionally semi-structured interviews (recorded) and a general survey were done with all students, including the non-participants.

Our approach to evaluation is considered a multi case holistic study in the blocked subject-project form [6]. The work is carried out in vivo. There are no "toy problems" involved and all "training" with the methods we use takes place on the students' actual projects. This adds a natural barrier to adoption, similar to that in other research environments where progress on the core work must be demonstrated. Our case study approach follows guidelines outlined in Kitchenham, L. Pickard and S. L. Pfleeger [7] who along with Basili [8] classify our type of approach as a formal experiment. The data collected from non-participants is used to cross the effect of our intervention.

IV. Examining the problem

Coding guidelines are taught to undergraduate students to encourage well-structured and consistent style. The importance of commenting is emphasised both as a way of enabling understanding in revision and of explanation for markers. Internal documentation in such work normally concentrates on *what the code is doing* [9, 10]. While useful for new programmers, understanding what code does is not the most valuable information a researcher can store either for themselves or for others. While a useful learning tool, the styles of documentation promoted to undergraduates fall short of our needs as researchers. A new approach is required to meet the needs of the research environment, and specifically the academic research environment that typically has smaller projects with fewer researchers.

In research the high value information is not what the code is doing. Other researchers should be able to eventually work this out from the code. Explaining what the code does can save time and should (at a high level of abstraction) still be documented, but if this is all a researcher does the most valuable part of their work – the research itself – is lost.

The question we believe researchers should address is *why something is being done.* A researcher's rationale and intentions are critically important. Another question is *why something is **not** being done*, and this relates to past efforts on this problem which may save another research months of work down a dead end. In a similar way ideas of changes that have been thought of but not carried through can help future work develop in a focussed manner that learns from past efforts even without the original researchers active involvement. This last answers the question of *what more could be done?* but also includes the original researchers thoughts on how to proceed.

Our current answer to capturing the information described here is to rely on publications. While reports and papers may capture some of the rationale decisions, they are unlikely to completely document them. Publications aim to focus on an interesting facet of the work. Another potential source of information would be research notes books, but these are private and unlikely to be made available to others. Both papers and journals as forms of external documentation suffer from the problem of the manual – most programmers will use them only as a last resort. The idea of extending internal documentation to additionally capture information relevant for research seems a natural progression of the existing use of documentation and a suitable replacement for the types of comments researchers may have found useful as undergraduates but now find a waste of time (often leading to minimal documentation or no documentation at all).

V. The design rationale

We see the capture of information about research work as critical to the maturing of computer science research, yet we recognise that on a practical level researchers need to benefit personally from their work (e.g. through publications) and support for documentation exists mostly at an abstract level and little is ever instantiated out side of publications. Put simply, an effort that has no benefit for the researcher but might benefit others in the field is likely to be very low priority for the only person who is able to do the work and as a result is unlikely to be adopted.

The use of coding guidelines for research provides a standard that is clear, and can be defended through an explanation of the benefits to the current researcher. The use of flexible *guidelines* rather than a stricter *standard* is intentional and designed to add agility and leave control with the individual researcher.

In our guidelines we specifically propose a documentation standard compatible with the dOxygen documentation tool. DOxygen extracts comments and produces html based documentation similar (but in our opinion superior) to JavaDoc. The use of such a tool combined with the capture of ideas in source code is designed to make capturing information as simple as possible without requiring the researcher to context switch or change tools away from their IDE or editor. It also makes the output easy to read and navigate, the conversion from input to output being automated and extracting further information from the source code itself.

In creating coding guidelines and environments for research we believe every effort should be made to reduce the adoption and usage barrier and to allow researchers to focus on their ideas, and not their tools. The guidelines presented here, though very generic, have shown high levels of adoption and satisfaction in our environment. We believe tailored guidelines, i.e. those made specific to a research domain and the common development environment of a research group in this domain, could further increase both adoption and efficiency. Such investigation was however beyond the scope of this research. The guidelines presented here are intended as a generic approach and were used on projects in a variety of research areas.

VI. The Research Coding Guidelines

In this section we provide a reduced version of the guidelines, the original is a 9 page document [11].

A. When efficiency is important

When time is being invested in highly efficient code, a similar effort should be invested in documenting why this method is needed (why is speed / disk space / memory conservation important at this point?) and how the method works.

The following guidelines are given for documenting highly efficient code:

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.

B. 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 and dOxygen /todo comment should be used to indicate future work.

C. When to comment

Take a half day each week 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 yourself and the code. Anything that isn't immediately clear and obvious to you now needs to be commented. If in doubt, ask yourself if your supervisor could understand this code without you there to explain it. Next think of questions they might ask and document the answers, particularly the rationale ones.

D. File Level Comments

Use these for tracking which files are yours for this project and which are being reused. If you change a file you have inherited or included from else where document this in a file level comment. File comments should also explain what the file is about and how it fits into the over all architecture (though in object oriented languages this may be relegated to the class level).

E. *Class Level Comments*

These describe the specific responsibilities of the class. What is it for? This comment should be general enough to cover all things related to the class, and specific enough to exclude the inclusion of functionality that belong in other classes. Decisions for the structure of the class including reasons why other options were rejected should be included. If the class structure is less than ideal an explanation of the problems and any ideas on for restructuring should be included in the comments.

In some cases details about methods or properties should be mentioned at class level (with further details against the item itself). In particular if a method or property appears not to fit the responsibilities of the class the reason for this deviation should be given. It might indicate a need to revisit the design. In some cases it may be appropriate to explain why a class exists at all, or what the class is supposed to be abstracting. It is particularly important that a link is provided in the comments between abstract research constructs and classes in the code that may not exactly map to these.

F. *Method Level / Function Level*

In most (but not all) cases 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 questions could be answered:

1) *Question relating to "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?

2) *Questions relating to "who" and "when"*

If the researcher is not the creator, who wrote? (e.g. note code form supervisor, for extracted from a paper etc) Even if the researcher coded it, if they were implementing someone else's algorithm details of the original source are important.

If this is an alternative implementation, this should be noted along with details on the original and why the change was needed.

3) *Questions relating to "how"*

What algorithm is being applied?
What data structure is being used?
Are there any fudges? (if so what?)

G. *Internal comments*

Internal comments should take only one line and should usually be document so they are not extracted by tools like dOxygen (introducing public variables is an exception where greater detail for extract may be required). Internal comments document how something is being achieved in a concrete fashion noting when each step occurs. They should be used sparingly and be as focussed as possible.

VII. Adoption Results

In addition to the coding guidelines we provided participants with a copy of dOxygen, an installation and setup guide and a config file with instructions. To help researchers accurately judge the cost/benefit of the tool in the second and third year of the experiment a sample input file and the generated output that went with it were provided. These files were from an MSc project in the first year of the experiment.

Our observation is that researchers liked the idea of dOxygen but until the cost and benefit were made clear were reluctant to invest time learning it which reduced take up in the first year. In the second year more students used it, but many only generated documentation at the end of their projects. In the third year with the sample input and output as well as the installation guide available earlier, more students decided to document in time to take part in a technical review. The technical reviews were largely based on the documentation. In their final surveys and interviews almost all students made reference to the documentation guidelines and or the dOxygen tool they used with it. All references were positive.

All large number of the more active participants from the final year have accepted funded PhD places and a number have commented that they will continue to use the tools and particularly the documentation guidelines for their PhD. With the experiment drawing to a close there was also concern expressed that this years MSc students will not have the benefit of the guidelines and tools. The department have responded and the resources from the experiment will be made available on the intranet. A number of PhD student have also expressed interest in trying the documentation guidelines, dOxygen and a technical review.

VIII. Researchers perception

As reflected by the success in adoption, researchers felt the coding guidelines were of benefit to them. In feedback one student noted how "it helps, as the project grows, to keep a clear vision of it" another said "advice on coding, backup, versioning issues etc are given without first having to ask the question". The hypotehsised benefits were realised, e.g. one student said they were helped by "code comments and the diary as a rough version of what I wrote in the final report" . A comment by a student in the final year on coding guidelines summed the benefit up quite well "It's easier to read a line (of comments) than to read ten lines of code, even if it's not difficult (code)" they said. Others expressed similar positive views to open questions on the benefits.

The Coding guidelines were ranking as the third most useful tool by students (out of 16 tools), behind only the "introduction to research" document and the webpage recommending tools students could consider. The sample dOxygen input and output (showing how comments following the guidelines are entered and how they appear in the generated output) was ranked the 6th most useful tool, compared to dOxygen itself which ranked 8th. The fact that dOxygen was used by the majority of the students and commented on very positively in most students feedback (to open questions) puts the relative rankings into some perspective. While the installation and basic setting guide to dOxygen was used by many students (in the second and third year) it ranked a poor 10th.

One observation was that documentation was no longer updated when researchers switched to report writing, concepts were instead embedded in the report. This shows a reluctance by researchers to context switch. Rather than indicating a problem we see this as validation for the idea that rationale of both software design decisions and of the research design decisions, questions, and ideas, should be placed in source code in a format that allows extraction and automatically generated documentation.

We noted some common problems in students use of the guidelines, these were corrected through discussion and caught in technical reviews. The problems included:

Under documenting – Often due to procrastination and a lack of a deadlines to increase the urgency of this task. The proposed solution was a recommendation to set aside half a day each week for catching up on documentation and other meta level work or small tasks.

Over documentation – This was a result of the wrong things being documented. It was necessary to remind researchers that at this level readers' proficiency in programming may be expected. Those who over documented tended to also under document their rationale.

Some students liked the idea of the guidelines and dOxygen so much that they tried to use both the guidelines and dOxygen with programming languages that were not supported by dOxygen. This met with mixed success in terms of the generated output, however students found the capturing of ideas to still be very useful.

Researchers found that advanced features of dOxygen were not needed to achieve reasonable benefits, yet learning extra features was of help e.g. \todo comments which generates an extra page in the document listing all the "to do" items with links to the places they occur.

Students found the installation guide, technical review and other supporting aspects related to both dOxygen and the coding guidelines useful. While the real benefit can be said to rest with the guidelines, other tools that lower the adoption barrier are vital if approaches like this are to be used.

IX. Engineers perception

From the Software Engineers perspective, the use of coding guidelines combined with dOxygen allowed a project to be reviewed in two hours with two hours spent in preparation by the reviewer. This allowed multiple reviews on projects in unrelated areas to take place on the same day and for a reviewer to extract the core design and research issues for a project with minimum effort. All reviews raised issues the researchers found significant. The coding guidelines made it particularly easy to understand and follow the development of a research project without requiring a high level of investment. The combination of code that followed the dOxygen created a standard platform that separated the review from the particular favour of operating system, IDE and to an extent programming language that the researchers chose to use (dOxygen was used for projects in C++ and Java successful and in other languages like C# with less success).

Technical review on the design of research in some cases caused the research direction to change. This in turn resulted in changes to the code. The documentation and in this regard be seen as a type of requirement. This is perhaps fairly unique to the research environment, where the problem itself can be changed to focus more on interesting issues that are discovered. The use of comments in source code to document the research ideas allows engineers to ask the right questions that can help researchers refine not only their ideas but also their focus. This sort of role outs the engineer in the position of a multi-disciplinary researcher and allows the connections between projects to be found and collaboration increased within a department.

Based on researchers ideas (as documented) and the structure of the code (as extracted and drawn up by dOxygen) it became possible for the engineer to comment on the design of the software architecture, modularization, and other key issues effecting the quality of the code. The engineers job becomes one of looking for mismatches and effectively prodding at them with questions for the researchers or suggestions of alternative approaches.

The coding guidelines and dOxygen output allow the engineering to focus at a high and abstract level, but if the guidelines are followed, also allows them to focus in on lower level abstractions and eventually code. This approach allows key aspects of the code to examined (including minor section that might otherwise be ignored) while avoiding areas that are not the focus of the research or may be built on inherited code that is out of scope.

From the Engineers point of view, the coding guidelines made an impossibly large task not only simpler but achievable with minimal investment.

X. Empirical difference

In Table 1 we show the combined results from the second and third years of our study. The table provides a comparison of the participants (21 members) and non-participants (28 members) group's average performance using the heuristic metric Δ, which we define as a students project mark, minus their course work mark. This can be considered their improvement.

Table 1 Summary data for 2004-2006

	Mean Project	STDEV Project	Mean Course Work	STDEV Course work	Δ	STDEV (Δ)
Participants	65.23	8.58	60.19	6.89	5.04	5.60
Non Participants	62.91	10.26	61.47	6.13	1.44	8.26
Change	2.32	-1.67	-1.28	0.76	3.60	-2.66

Participants had a higher degree of improvement. Coding guidelines were one of many changes and as such represent only a part of the treatment applied, but with both observations and surveys indicating their importance they are thought to have played a significant part in these results.

XI. Future work

The ability to enter comments directly in the code is a key feature of our approach; however a software tool to review the comments and allow them to be edited would be welcome for those occasions when researches do take time out just to document. Something like dOxygen, but with editable fields that are then written back into the source code would be idea. As an extension to such a tool it might be possible to add meta information listing the question that are being answered or additional information and storing some of this in an XML or similar format in a separate file. Further extensions could allow comments and questions by the engineers and multiple other “reviewers” to be added and stored outside the code. We are only beginning to examine the question of improving the research software development environment, and there is a lot more that can be done.

On the engineers side we have observed the benefits of the documentation standard, however a systematic study would be beneficial. Such a study might examine the quality of feedback with varying amounts of support from engineers with source code only, traditionally documented code, and code documented according to these guidelines. We would expect that the better documented the code is, the higher the quality of the feedback would be. A tool as described above would greatly help with cleaning up comments.

XII. Conclusion

We believe the coding guidelines presented here are of benefit to researchers of all levels of experience and could greatly improve the quality of the information we have about research in computer science. They capture the high value information that would otherwise be lost and avoid the problem of task switching that other forms of documentation create. Combined with technical reviews they increase the flow of information and improve the quality of research code and on occasion the quality of the research work itself.

The guidelines are an agile approach that allow research projects to have their ideas and requirements captures on the fly as they occur. They provide a systematic approach that can be relatively consistent across projects and researchers, greatly easing the task of engineers. While benefiting the individual researchers, the guidelines allow improved communication and reduce the risk of key ideas being lost.

The guidelines are a useful start to improving the research environment in a way that benefits both the original researchers and the field as a whole.

XIII. References

[1] A. Oboler, D. M. Squire, and K. B. Korb, "Software Engineering for Computer Science Research - Facilitating Improved Research Outcomes," *International Journal of Computer and Information Science*, vol. 5, pp. 24-34, 2004.

[2] A. Oboler, "Examining the use of Software Engineering by Computer Science Researchers," presented at In Proceedings of Education Students' Third Regional Research Conference, Graduate School in Humanities University of Cape Town, Cape Town, South Africa, 2003.

[3] OECD, *The Measurement of Scientific and Technological Activities - Frascati Manual 2002 : Proposed Standard Practice for Surveys on Research and Experimental Development* Frascati, France: Organisation for Economic Co-operation and Development, 2002.

[4] R. L. Glass, "Searching for the Holy Grail of Software Engineering," *Communications of the ACM*, vol. 45, 2002.

[5] D. Howe, *The Free On-line Dictionary of Computing* 2005.

[6] R. K. Yin, *Case Study Research: Design and Methods 2nd Edition.* Beverly Hills, CA: Sage Publishing, 1994.

[7] B. Kitchenham, L. Pickard, and S. Pfleeger, "Case Studies for Method and Tool Evaluation," in *IEEE Software*, July ed, 1995, pp. 52-62.

[8] V. R. Basili, "The role of experimentation in software engineering: past, current, and future," presented at Proceedings of the 18th international conference on Software engineering, 1996.

[9] Sun Microsystems, "How to Write Doc Comments for the Javadoc Tool," 2000-2004.

[10] J. K. Nair, "Computer Program Documentation Standards: Version 1.3 " in *Education Infrastructure Project*, vol. 2006: Virginia Tech CS Dept, 1998.

[11] A. Oboler, "RAISER Coding Practise," Lancaster University: SERE Project Resource, 2004. http://www.comp.lancs.ac.uk/computing/users/oboler/RCD.doc