Formal Technical Reviews for Research Projects

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

Formal Technical Reviews (FTRs), where a software developer and a team of reviewers walk through a piece of code to assess its quality, have been used in the computing industry for many years. In this paper we look at the suitability of FTRs in an academic research setting.

We investigated the use of FTRs combined with other approaches in a set of case studies over a three years period. MSc students were used as test subjects with their dissertation projects being used as the research products. In all about half the student volunteered to participate each year, though very few took part in FTRs initially. Changes to the experimental setup through the introduction of additional artefacts to lower adoption slowly increased the participation in FTRs.

Our initial intervention included the introduction of a documentation style for software coding, this was largely about the approach to commenting and the specific type of data that should be captured for a research project. We also introduced the use of a software tool to extract comments from computer source code and generate a set of hypertext documentation complete with reverse engineered design diagrams. The generated documentation aimed to facilitate both the FTRs and discussion between researchers, for example office colleges, supervisors, or researchers at remote locations. We also found it necessary to create an installation and configuration guide to lower the barriers to adoption. A side effect of the generated documentation was the ability of the software to create a website for the participants with very little manual effort. This increased the benefits associated with the cost of learning the tool, and effectively reduced the barrier to FTR participation by making the tool more obviously cost effective.

While computer science researchers in universities typically shy away from quality improvement processes and have a high level of resistance to the introduction of new tools and techniques, the combination of approaches we used is shown to reduce the effort of participation required by both researchers and review participants to acceptable levels. This level of acceptance increased as the experiment progressed and more of the tools became available, and could with future work be improved further.

Past research shows a low level of adoption of similar software engineering approaches, and the current research repeatedly highlights the initial scepticism of both students and supervisors, however our data suggests that these barriers can be overcome with the right combination of tools and this can lead to improved collaboration between researchers in computer science or related areas.

I. INTRODUCTION TO TECHNICAL REVIEWS

Technical Reviews are the most widely used approach in Software Development to validate the quality of a product or process [1]. They involve expert inspection in order to provide feedback that enables improvement. There is a large body of literature within the area of Technical Reviews (and more specifically on Code Inspection) but it has often been hard to reconcile and consolidate this information due to its volume [2].

Johnson [3] 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." [1] 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" [3]. The outcome of a review is therefore not only constructive but concrete.

What goes into a technical review? Sommerville explains that "reviews 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" [1]. Though specific to software each of these items has a more generic nature in research. For example test plans here refer to the documented test cases to evaluate whether a piece of code is functioning correctly. A test plan could equally be devised to ensure that a survey is understood and completes as the experimenter would expect (e.g. a document explaining how trial will be run could constitute a test plan). Likewise the configuration of a software tool used to aid the research could be seen as configuration management and once documented could be reviewed to highlight and validate any assumptions. By taking these definitions loosely Formal Technical Reviews can be seen to apply to any process or product created in a systematic way.

Formal technical reviews involve experts. More specifically they involve a producer (author of the work being reviewed) who states when their work is ready for review, and one or more reviewers. The reviewers "prepare" for the review by analysing the code, product, or process documentation, and taking notes. The producer and reviewer(s) then meet and go through the item under review and the recommendations in a way that is not critical of the producer, but points out weaknesses in the item.

The nature of a review depends on its purpose and when it takes place. For code reviews this depends greatly on the position in the Software Development Life Cycle. Reviews can be described in terms of there three orthogonal dimensions; Objective, Interaction Mode and Technique [4]. 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). The interaction mode is either synchronous (working together at the same time) or asynchronous (working at differing times). Today with the availability of collaborative work methods both synchronous and asynchronous approaches can be used whether participants are physically located together or not. The Techniques dimension can range from free review through to a detailed checklists of items. It specifies the strategy to be used by the review team and what sort of items are being targeted in the review.

Like other Software Engineering approach, what is taught to undergraduates is not what is practiced by researchers themselves [5], we discuss this with particular reference to technical reviews in section II.

II. USE IN RESEARCH

In a year long research project during 2002 we examined the Use of Software Engineering in the Research Environment [6]. This examination included web-based surveys for American and later Australian academics.

Table I shows the levels of use of Technical Reviews by the Australian sample. Table II shows the level of use of technical reviews by the American sample. The surveys were introduced though e-mails sent to department heads in all computer science (or similar) departments in each country. They e-mails asked the department heads to forward the survey's web address and a short explanation (provided) on to their staff. The survey to the USA was released first.

In feedback received via reply e-mails it became clear that some heads of departments in the United States saw Software Engineering as something only relevant to those staff who taught Software Engineering courses. As a result of the selection made by department heads, and self selection made for similar reasons, in the American sample 72% of respondents had taught software engineering. A few months later we released the Australian survey after removing reference to software engineering from the introductory e-mail and the survey itself. In the Australian sample only 43% of respondents had taught software engineering.

More surprising than the level of Software Engineering Educators in the sample was the difference in seniority and experience between the two samples. In the American sample 14 people (over half the sample: 56%) had 20 or more years professional experience in software development / software research, only 3 people (12%) in this sample had less than 5 years experience. This compares with the Australian sample where 16 people (46%) had five year or less experience, and only 6 people (17%) had 20 or more years experience. These were very different samples.

TABLE I Use of FTRs by Australian Academics

	Don't Know	Don't Use	Use Rarely	Use Sometimes	Use Often	Use Always
N	15	11	1	6	1	1
%	43	31	3	17	3	3

TABLE II Use of FTRs by American Academics

	Don't Know	Don't Use	Use Rarely	Use Sometimes	Use Often	Use Always
N	5	2	3	6	8	1
%	20	8	12	24	32	4

As the tables show, the American sample (Table II) with its higher level of engagement in software engineering teaching and higher level of experience had less people who did not know what an FTR was, and more people who made use of it.

The Australian sample in Table I showed both less awareness (expected in a less specialized sample) but also a far higher level of people who knew what it was, but chose not to use it. This is possibly a reflection of the higher number of people without experience who have been taught about technical reviews but due to their lack of experience outside academia (and the lack of application of FTRs inside academia) have not used them.

III. EXPERIMENTAL OUTLINE

Building on our 2002 work, our current research sought to find ways of integrating Software Engineering into the research environment in a way that would provide cost effective benefits. The main goals of Software Engineering in the Research Environment would be improved collaboration during projects (partly demonstrated through technical reviews) and improved knowledge capture between projects as a result of better and more complete documentation of research related information [7].

In our three year experiment we offered Advanced Computer Science Masters students the chance to participate in an experiment called SERE (Software Engineering in the Research Environment). Student who chose to participate were given access to artifacts, tools and meetings (including technical reviews) to assist them during their five month research projects. Using a new process view of research [8], we introduced a number of guidelines and recommended tools, but left students free both to choose which tools and approaches they adopted, and to how they adapted these to their own styles and needs. Mostly of the tools provided were from Open

Source projects allowing students free access both at university and at home.

The nature of the experiment was introduced to all students in an initial lecture as part of the regular teaching schedule. Participation was optional but required registration after the initial lecture. The experiment then proceed to use a case study based approach to enable triangulation both to validate and to allow deeper understanding of students comments. The case studies took the form of a multi case holistic study in the blocked subject-project form [9]. We following guidelines on case studies for software engineering from Kitchenham, L. Pickard and S. L. Pfleeger [10] who along with Basili [11] classify our type of approach as a formal experiment.

In total over the three years of the experiment 37 students opted in (our participants), and 42 students opted out (non-participants). The first year saw the participants (the fourteen volunteers) split into control and experimental groups, each with seven students in them. Both groups had access to a website with tools, but only the experimental group had access to meetings – this included the Formal Technical Reviews. The fifteen non-participants did not have access to the website. This design was created to remove sample bias, however this was later seen to be inadvisable as other factors played a far larger part than the potential for experimental sample bias.

Over all three years the experiment was introduced in an initial lecture which included a survey on students' background and a form granting permission to access further information as needed and explaining the anonymous nature of any data released publicly. All students were also asked to complete a final survey as they were finishing the project (or shortly thereafter).

The idea of the control and experimental group from within the volunteers was dropped after the first year mainly due to ethical considerations: Control Group members felt they had significantly missed out and were unable to take full advantage without the additional assistance given to experimental group members. While such an intervention based on the researchers self selection was considered problematic, allowing self selection and offering equal involvement to all volunteers was considered the only acceptable path given the impact students felt the experiment was having.

In the later two years the two groups were simply volunteers and non volunteers. Data was therefore requested from all students regardless of whether they volunteered. The sample size was 23 participants and 27 non-participants over the final two years. The second and third years also saw extensive use of interviews, which were recorded on a dictaphone and later transcribed. Other types of meetings, including technical reviews with participants and post project used to augment the final surveys with both participants and non participants were also recorded.

The data collected at the end of the project reflected on the student's choices about participation, the spread of knowledge between participants and non-participants (practically none), and students view of their completed work. Participants were asked to complete an additional survey on the experimental approaches used (including Formal Technical Reviews).

Each year the approaches were improved in an evolutionary manner. In the case of Formal Technical Reviews this meant moving the time of the review to earlier in the process (after the first year), and then providing preparation guidelines before the reviews in the third year. The need to encourage students to hold the review when the work was only partially completed, rather than putting it off until the work was "closer to finished" became evident after the second year.

IV. Using the Reviews

Formal Technical Reviews were offered in all three years but the first year saw students constantly putting off the reviews until later. These reviews were intended purely as code reviews.

In the second year students' code and their documentation, i.e. internal commenting in the code, were review indirectly through the output generated by the dOxygen software package. The dOxygen tool is an open source program that creates diagrams of the stricture of a piece of code, and extracts commented the programmer writes into the code, presenting it in a structures format. The use of an installation guide providing a common configuration (which students then altered slightly to fit their needs) provided a consistent style of document to review. The coding guidelines provided recommendation on both how and what students should be recording in their comments. Contradictory statements, gaps in the explanation and a lack of documented rational were the most common types of problems picked up. These at times lead the student to consider larger problems of which these were only symptoms.

In the third year two reviews were offered. The first was on the code (via the dOxygen document about it) as per the second year. The second looked at the code, but also the process plan, a new approach we created for documenting the student's personalised approach to their research project using code comments and dOxygen. We used the familiar structure of programming code to lend systematic structure to a plan of the research. We reused dOxygen as it provided a nice readable format and allows systematic navigation through the output. As the process model was stored as just another file of source code, it became possible to integrate the documentation of the process with the documentation of the product. This allowed a more detailed review focussing on the research ideas at both the local and more global and abstract levels.

One of the participants (Case 0629) in a Technical Review told us they had a learning difficulty that affected their memory of sequential reasoning. The student suggested the process plan would help them with this difficulty, immediately after the tool was presented to them they said "that is a brilliant idea, I love it. I really do." In the post project interview they said the idea of documenting the process was the more useful thing they got from participation (i.e. more useful than Technical Reviews) they added "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."

It's possible that given more time more regular reviews of the process plan would increase this benefit even further by allowing other researchers (acting as reviewers) to add their thoughts and concerns about the research. This was seen to some extend in one review which was attended by a student's supervisor and then fed in to their later supervision meetings. The supervisor suggested additional improvements including the addition of change tracking for the process plan, something we later implemented. The change tracking created a version of the process plan where any deleted information was struck through and written in bright green, while new information was highlighted in yellow. In two other cases the review involved a non participating observer (to inspect the review for experimental purposes). While not contribution in front of the student, these researchers (both PhD students) highlighted what they saw as the main issues in the students project in a debriefing session held immediately after the student left.

Perhaps the most interesting fact about the reviews was that despite the large impact and the level of thought they provoked in the students, a review took only two hours to prepare. This was done using a re-engineering pattern we created called "Read Everything in Two Hours" [7]. It is based on the use of the guidelines and generated documentation to stream line the reviewing process. Without knowing much about the students projects, we have shown that another researcher can significantly contribute in a detailed way without a huge investment of their time. As Technical reviews allow the reviewers to get third parties' opinions on either the whole artifact under review, or components of it that the third party may have expertise in, it should be possible to extend this two hours by adding a second stage of preparation where additional opinions are sought on specific areas. While working for module code, this idea may be even more relevant for reviews of research artifacts that do not involve code, for example in our process plans where a documented experimental approach could be considered in isolation by someone with experience of similar work..

V. QUANTITATIVE RESULTS

While shown to be easy to implement we have yet to show the value of the reviews. In this section we consider the impact of experimental participation, and specifically of technical review participation from a quantitative approach. The results based on this data one are inconclusive, but point in the right direction. In the next section we add qualitative data to our considerations.

One source of independent data on the impact of our interventions was student's marks. Specifically we examined a students project mark in comparison to the average coursework mark for that same student. This degree of change value (?), would be largely positive if students project marks exceeded their coursework mark, and would indicated a student performing above their expected ability based on all other information available about their performance. We define ? as the student's project mark, less their course work mark and consider it to be a metric of improvement.

Some improvement is expected across the board as students will likely have focused on a project that reflects their strengths. By comparing the average coursework mark of each group we can check that the samples have similar starting abilities. By comparing the average ? of participants compared to the average delta of non participants we can see if one group improved at a different rate to the other.

The comparison of project, coursework, and ? values is shown in Table III from the results of students in the second and third cycle of our study. The first cycle are not included as many of the interventions did not exist and the approach is not directly comparable. The table compares 23 participants with the 27 non-participants based on their group's average performance using the ? metric.

Table III Summary data for 2004-2006

	Mean Project	STDEV Project	Mean Course Work	STDEV Course work	?	STDEV (?)
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

Table III shows participants had a higher degree of improvement than non participants. Technical Reviews were only one aspect of participation, and indeed only four of the student undertook a Formal Technical Review, though others received informal feedback of a similar but far less detailed nature. The average delta for the four review participants was 4.7, and the weakest student in this group had the largest delta (8.97). All the students who took technical reviews however performed above the average (for participants) in both their project marks and coursework marks.

The sample size is too small to draw conclusions beyond saying they do not look like exceptions to the general pattern of those who participated. To learn more about the impact of Formal Technical Reviews a far larger sample would be needed, however this would be difficult to obtain unless systematically integrated into the project experience by supervisors for half the students.

Another approach is to examine the perceived impact, i.e. students own thoughts on the value of Formal Technical Reviews. In post project surveys participants were asked to rate the various tools offered to them on a scale of 1 to 5 with the values: 1 useless, 2 some use, 3 useful, 4 very useful, 5 the most useful bit. Students also had the option of not answering the question. Three of the four users of Technical Reviews returned these final surveys. Two rated Formal Technical Reviews as "the most useful bit" and the other rated it "very useful". This is out of a list of 16 interventions in the second year with an additional 3 for those participants in the third. This shows a very high degree of value placed on technical reviews. While this data is still from a small sample it is amazingly consistent. 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.

VI. QUALITATIVE RESULTS

In this section we discuss the technical reviews as students saw them. We provide the reflections of those who took part, and the reasons why others chose to miss out on the opportunity.

We begin by considering Case 0417 from the first cycle. This student was given a final reminder about booking an FTR one month before the project was due. The student's response said "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 making it hard for others to reuse their work. This was not an isolated case and provided part of the reasoning for moving Technical reviews to the start of the coding period. IT is also why we emphasised that students were not expected to have finished the coding before having their review.

The observer in a cycle two Technical Review (Case 0526) was asked what he thought of the review. "It was a good work through. You did quite a lot of going through his problems, and came up with solutions" the observer said. He pulled out 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 "over all very informative" to the student.

In the review itself (again case 0526) the lack of documentation was stressed, and the student agreed that with more document "the good thing is that in the end you have pretty good documented code" though they added that "for me what I have now is sufficient for me to look back and understand." A to-do note left in the code was questioned. The student's reply was that "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 names 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. They had not generated dOxygen themselves but found it useful. Over all 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 ended with the student being asked if he felt the review had been worth while, he 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 is more pressure. It's a small thing, but even I might say oh my, it's another thing."

One Formal Technical Review 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. This shows the value of the review in focusing discussion and highlighting issues that might otherwise not be discussed. 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."

In their post project interview one 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. Let's 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." The problem the research recognized also holds true when new researchers need to be introduced to the work..

In a review of case 0629 the observer was asked how they thought the student found the FTR, they 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 more." In contrast, Case 0525 involved a strong student who chose not to participate. They were asked after the project whether with hindsight they would have been involved, 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". While there are benefits, particularly in the area of collaboration, not all researchers wish to open their work up to detailed scrutiny.

VII. FUTURE WORK

The use of process plans has only been trialed once. The individual process plans were all created by adapting a generic template. In a larger trials it may be possible to create more specialized template for different research areas and then use Formal Technical Reviews to compare how these plans diverge. This would expose researchers to other approaches and other ideas from people working along similar lines to themselves.

Another possibility is to extend the range of artefacts that are reviewed. Having used computer based tools for qualitative research, it seems the configuration and design choices there closely mirror the situation in software development. Technical reviews could therefore be adapted to wider use as the users of software tools are today more expert and in ever greater control of their software.

VIII. CONCLUSION

In this paper we show the comparative lack of use of Formal Technical Reviews in the academic research environment compared to industry. We show that Technical Reviews and other Software Engineering approaches can have positive benefits in computer science research, and suggest that Technical Reviews particularly could be adapted to other research environments that make use of software which the researcher configures. We suggest that in the research environment the main benefit of a technical review is the pooling of expertise and the ability to invest a small amount of time to learn about another research project which may lead not only to a sharing of ideas, but perhaps also to future collaborations.

IX. REFERENCES

[1] Sommerville, I., *Software Engineering, 6th edition*. 2001, Harlow, UK: Addison-Wesley.

[2] Laitenberger, O. and J.-M. DeBaud, *An Encompassing Life-Cycle Centric Survey of Software Inspection*. 1998, Fraunhofer Institute for Experimental Software Engineering: Germany.

[3] Johnson, P., *The WWW Formal Technical Review Archive*. 1999, University of Hawaii.

[4] Tjahjono, D., *Evaluating the Cost-Effectiveness of Formal Technical Review Factors* in *Department of Information and Computer Sciences*. 1994, University of Hawaii: Honolulu.

[5] Oboler, A., D.M. Squire, and K.B. Korb. *Why don't we practice what we teach? Engineering Software for Computer Science Research in Academia*. in *In Proceedings of the First International Conference on Software Engineering Research and Applications*. 2003. San Francisco, USA.

[6] Oboler, A., *USE CSR*, in *School of Computer Science and Software Engineering*. 2002, Monash University.

[7] Oboler, A. and I. Sommerville, *Research Documentation Guidelines: Capturing knowledge, improving research*, in *Fourth International Conference on Information Technology : New Generations (ITNG 2007)*. 2007, IEEE Computer Society: Las Vegas, Nevada, USA.

[8] Oboler, A., I. Sommerville, and S. Lock, *Reflection: Improving research through knowledge transfer*, in *The First International Conference on Software Engineering Advances (ICSEA 2006)*,. 2006, IEEE Press: Tahiti, French Polynesia.

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

[10] Kitchenham, B., L. Pickard, and S. Pfleeger, *Case Studies for Method and Tool Evaluation*, in *IEEE Software*. 1995. p. 52-62.

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