Software Inspections, Reviews & Walkthroughs

Marcus Ciolkowski
University of Kaiserslautern &
Fraunhofer IESE
Kaiserslautern, Germany

Oliver Laitenberger
Fraunhofer IESE, Kaiserslautern, Germany

Dieter Rombach
University of Kaiserslautern &
Fraunhofer IESE, Kaiserslautern, Germany

Forrest Shull
Fraunhofer Center Maryland,
College Park, MD, USA

Dewayne Perry
University of Texas, Austin, Tx, USA

1. INTRODUCTION

While software has become one of the most valuable products of the past decades, its growing complexity and size is responsible for making it one of the most challenging ones to build and maintain. The challenge stems from the fact that software development belongs to the most labor- and, at the same time, knowledge-intensive processes of today's world. The heavy dependence on knowledgeable human beings may be one reason why software development is often compared to an art or craft rather than to an engineering discipline. However, it has almost become impossible nowadays for a craftsman to produce large software systems according to a given schedule, to a limited budget, and to the quality requirements of a customer at delivery. Hence, researchers as well as practitioners are increasingly obliged to address the question of how to integrate engineering principles into software development. An important one is to perform quality-enhancing activities as early as possible. Despite the simplicity of this principle one can observe in the software industry that the activity of detecting and correcting software problems is often deferred until late in the project.

To address this issue, engineering-oriented software organizations have started to implement rigorous inspections, reviews and/or walkthroughs (in this paper all referred to as "inspections"). But still, a large number of organizations do not take full advantage of these approaches, which prevents them to base their software development approach on engineering grounds.
The main objective of the IMPACT project in the area of software inspection is to collect demonstrated success cases, perform root cause analyses as to what contributed to the success cases in terms of research and transfer activities in software engineering, and derive lessons learned to maximize the success in other interested organizations. The research results in the inspection context include both new techniques, methods and tools as well as sound empirical evidence regarding the effectiveness and context dependency of inspections. The results show the importance of methodological and empirical software engineering research.

Empirical software engineering has lead to overcome the ‚factoid' that testing is the most effective defect finding technique, and helped maturing software development in practice one step further towards an engineering discipline.

This abstract first presents some of the history of inspections, walkthroughs and reviews. An example is briefly described to illustrate how research impacted industrial software development practice in this area. Finally, challenges and questions as well as areas for further work are outlined.

2. HISTORY

This historical overview is based on material of Tom Gilb. He therefore earns the credit for this part.

Walkthroughs were widely practiced before inspection at IBM. They were conducted by someone presenting the entire logic of an artifact and paraphrasing it aloud, while others listened or asked questions. Walkthroughs primarily aim at training and only secondarily detecting or measuring defects. In a direct comparison a British IBM Lab showed that inspection was an order of magnitude better at finding defects than structured walkthrough.

As the 1960s drew to a close, it became apparent that delivering defect free software on time was difficult. IBM was one of the largest software houses in the world. There were a number of streams of development of better quality control methods for software, at about the same time.

The major players were Michael Fagan, Harlan Mills and Ronald Radice [2], with Watts Humphrey supporting Fagan and Radice in their developments. Mills was in the Federal Systems Division, outside of Humphrey's domain. But Mike Fagan worked directly with Mills for a few years. Mills also was one of the first working on reading techniques

Mills and his associates packaged inspection into a larger attack on the software quality and time problem known as the Cleanroom method [4]. There were a number of components to that such as structured programming, user profile testing, evolutionary project management, design reviews, and code inspections including "reading by stepwise abstraction".

In parallel with Fagan, Ronald Radice, at Kingston Labs used the inspection process in 1975 for levels of specification above pseudocode. Some of his ideas went into the development of the Capability Maturity Model (CMM). Level Three of that model was called 'Peer Reviews'. Many other elements from the practice of inspections would influence other levels of the CMM.

3. RESEARCH IMPACT ON PRACTICE

As part of the IMPACT initiative several industrial inspection implementations were analyzed to determine how research has influenced the industry practice. Among those, the NASA/SEL is presented here as an example to illustrate the analysis principle.

The NASA/SEL-Example

Some of the earliest work in the area of inspections that influenced the work at NASA/SEL can be traced to Hetzel [1] and Myers [5], who performed studies of developers that showed there was little difference between the effectiveness of code inspection and testing for finding defects.

In the late 1980's, as part of ongoing efforts to improve software product quality at NASA's Goddard Space Flight Center (GSFC), an investigation was undertaken in a research setting to explore the effects of different test techniques (structural and functional testing) on defect detection. Somewhat surprisingly, the results indicated that the initial results of Hetzel and Myers did not seem to hold. In fact, several distinct benefits to code inspection were identified:

- Code inspection was significantly more effective than either functional testing or structural testing for finding defects.
- Code inspection led to better estimates of code quality.
- Code inspection found different fault types than testing.

Based on these research results, code inspection was introduced at NASA GSFC for development use. However, when the defect detection rates were compared to historical baselines, the results were disappointing: code inspection seemed to have very little effect on defect detection.

Researchers attempted to analyze the reasons for this result and hypothesized that: (1) inspectors may require specific techniques for finding defects in order to inspect effectively, and (2) inspectors will inspect less effectively if they know they can count on downstream testing of the software to catch what they miss.

Based on these results, software reading (i.e. an inspector's individual preparation strategy for finding defects) was hypothesized to be an important component of inspection effectiveness. To test this, the "Cleanroom" process was formulated. Cleanroom provides developers with a specific reading technique (in this case, developers were asked to use the step-wise abstraction process [3]) and a motivation for reading (the developer is asked to certify quality without being able to test the code). In this way, code inspections were again moved to industry in the form of the Cleanroom process, applied on a project at NASA GSFC. It was shown effective in an initial study, in which it was shown that Cleanroom reduced the failure rate during test by 25% and at the same time increased productivity by 30%, mostly due to reducing the rework effort.

Although the Cleanroom experiments were successful, it was recognized that one necessary direction for further work was finding reading techniques that could be applied for inspections of other artifacts, such as requirements or design. As a result, the basic research results concerning reading techniques have been tailored for and applied in a number of different environments, for various goals. Among them are Defect-Based Reading (DBR), Perspective-Based Reading (PBR), Object-Oriented Reading Techniques (OORTs), and Use-Based Reading (UBR). DBR is a family of reading techniques for defect detection in requirements expressed using a state machine notation called Software Cost Reduction. PBR is a family of reading techniques focused on defect detection in requirements expressed in natural language; further, PBR has been tailored for design and code documents. OORTs are another family of reading techniques designed for inspections of high-level designs. Finally, UBR is a family of reading techniques focused on fault detection in user interfaces.

4. CHALLENGES AND QUESTIONS

Despite the large volume of published inspection material, there are some important challenges and questions underlying this work. First, a sound examination of the impact that research on inspections did have on practice requires more insight into a company. This is usually beyond the scope published in research papers. Hence, we challenge the members of the software engineering community to get us access to companies that are willing to provide this information in the form of success stories. If possible, the success stories should have some quantitative underpinning.

5. FURTHER WORK

The results from the Impact project also revealed areas for further work in inspection. Among others, further work requires a better integration of inspections in the overall software development process, the clarification of its relationship with other defect analysis techniques such as testing, verification, or model checking and the clarification of its relationship with construction techniques such as design documentation or languages.

6. SUMMARY

This paper presented some information on the impact initiative in the area of walkthroughs, inspections, and reviews. It consisted of several parts. The first introduced some historical information. It showed that inspections are related to research efforts back in the 70's. It also presents an example success story together with a succinct description of how research influenced the practice. The success story was the result of researchers and practitioners working closely together. As a summary statement one can say that in the inspection area, research did have and still has impact on the industrial practice.

Most of the inspection work can be traced back to the original publication of Michael Fagan. Although he is mainly mentioned as the "inventor" of inspections, the historical overview revealed that other researchers did participate in the development of inspections. Since then, many others fine-tuned the approach to adjust it to the specificities of today's software development approaches. In this way, research and practice goes hand in hand to come up with new solutions. It is also fortunate that large communities, such as the ISERN-community, selected inspection technologies for the purpose of progressing empirical work in software engineering. Based on their findings, myths can be examined and funding can be directed to the most valuable areas.

7. REFERENCES

[1] Hetzel, W. C., 1976. An Experimental Analysis of Program Verification Methods. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science.

[2] Kohli , O. Robert, and Ronald A. Radice, Low Level Design Inspection Specification, TR 21.629, IBM System Communications Division, Kingston NY 12401

[3] Linger, R. C., Mills, H. D., and Witt, B. I., 1979. Structured Programming: Theory and Practice. Addison-Wesley Publishing Company.

[4] Harlan Mills, "Cleanroom Engineering", American Programmer, Pages 31-37, May 1991

[5] Myers, G. J., 1978. A controlled experiment in program testing and code walkthroughs/inspections. Communications of the ACM, 21(9): 760-768.