Empirical Software Engineering, 6, 81–84, 2001.
© 2001 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Cost-effective Detection of Software Defects through Perspective-based Inspections

OLIVER LAITENBERGER* Oliver.Laitenberger@iese.fhg.de
Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D-67661 Kaiserslautern, Germany

Keywords: Quality assurance, object-orientation, perspective-based inspection, architecture-centric inspection organization, experimentation, replication and meta-analysis

Software organizations must deliver high quality products on time and within budget to remain competitive in the marketplace. However, the reliable and predictable development of high quality software continues to be a major problem, largely due to the inadequate and late removal of defects.

One of the proposed solutions for early detection and removal of defects is software inspection (Fagan, 1976), (Gilb and Graham, 1993). A software inspection requires a team of qualified persons to scrutinize software documents for defects. It applies to any development methodology and to any phase in the software development process. The value of an inspection stems from the fact that it improves quality and saves defect costs, since it minimizes the time between insertion of a defect and its discovery. This is important because the cost of finding and fixing a defect increases significantly each time it propagates to the next development phase. To fully achieve the expected benefits, two practical questions related to software inspection need to be answered: how to organize a particular inspection in a development project with a large volume of documentation and how to provide technical support for the defect detection activity of an inspection with reading techniques (Basili, 1997).

This research presents a new strategy for inspection organization. The basis for this strategy is the distinction between logical entities and their documentation. This distinction is necessary because, in contrast to other engineering disciplines, the entities in software engineering are invisible and cannot be inspected per se. It is only the documentation of the entities that can be scrutinized for defects. The strategy, dubbed architecture-centric inspection organization, suggests partitioning and grouping the documentation to be inspected according to logical entities rather than characteristics of the documentation, such as their type or size. Following this strategy, the most appropriate unit to inspect can be determined. This unit refers to documentation that contains relevant information about one or several logical entities. Organizing an inspection in this manner avoids a decline in the cost-effectiveness of inspection because of fatigue effects or loss of motivation. Although the architecture-centric approach is generally applicable, it is particularly valuable

* PhD thesis submitted and accepted at the University of Kaiserslautern. Advisors: Prof. Dr. Dieter Rombach, Prof. Dr. Victor Basili

for inspections in object-oriented development projects with its graphical diagrams and delocalized information.

To provide systematic support for detecting defects in the documentation of any logical entity, this research introduces the Perspective-based Reading (PBR) technique. The basic idea underlying PBR is to focus each inspector on a specific viewpoint and to provide active guidelines in the form of operational scenarios for scrutinizing the documentation of one or several logical entities for defects. To elaborate this idea into a generally applicable technique, the work provides an in-depth description of PBR. Moreover, it explains how to tailor PBR to various inspection situations. Because of the specific focus and the procedural guidelines, the PBR technique is hypothesized to increase the (cost-)effectiveness of inspections in comparison with traditional reading approaches.

To quantify the potential benefits of PBR in conjunction with the architecture-centric inspection organization, three empirical studies were conducted. The primary goal of these studies was the evaluation of PBR in comparison with more traditional reading techniques, that is, ad-hoc or checklist-based reading. The studies were performed in different environments using industry practitioners with various backgrounds. The studies were organized as controlled experiments and quasi-experiments, since these two study types allowed the examination of cause-effect relationships. The results of the studies demonstrated the value of PBR in terms of a higher (cost-)effectiveness when compared to more traditional reading techniques, such as ad-hoc or checklist-based reading. Moreover, the results indicate that PBR helps reduce the impact of human experience on inspection results. Hence, given the current weight of evidence from the three empirical studies, there is empirical support for the assumption of this research work that PBR outperforms traditional forms of reading.

Since there is little practical experience in the software engineering domain relating to the various possible types of studies, their design, and their analysis, the lessons learned about experimentation can be considered as a separate contribution of this work.

As mentioned above, controlled experiments as well as quasi-experiments were conducted for validating the hypotheses. While there are already experiences with controlled experiments, the use of quasi-experiments for validating hypotheses has so far not been reported in a software engineering context. Quasi-experiments are used instead of controlled experiments when random assignment is impossible or when, for practical reasons, it is necessary to use pre-existing or naturally occurring groups. This situation can often be found in a software engineering context.

The research also discusses the issues related to the design of (quasi-)experiments. Within-subject designs are elaborated upon in more detail. In a software engineering context the benefits of within-subject designs usually outweigh their disadvantages. Hence, researchers should look more often at this type of design when planning and conducting empirical research.

Finally, empirical studies in software engineering are often performed with a low number of subjects. The small sample size often results in studies with low power. Low power is considered a potentially strong contributor for not finding statistical significance, making it difficult to interpret empirical results. One possibility for tackling the problem of low power is to replicate an empirical study. However, replication raises the question of how to compare and combine the results of the original study and the replications. In this research meta-analysis techniques were applied and found to be a useful tool for the combination of empirical results. Although there is currently a debate about whether meta-analysis techniques are applicable in a software engineering context, other researchers should consider these techniques in their arsenal of analysis approaches.

The main results of this research can be summarized as follows. By providing a detailed explanation of the architecture-centric approach for inspection organization as well as the Perspective-based reading technique, researchers will be able to extend the scope of the approaches, perform replications of the empirical studies, and set up new ones. The reported experiences gathered in the design, execution, and analysis of the empirical studies may be helpful for these purposes. However, the lessons learned in those studies are beneficial for the empirical software engineering field in general. Practitioners on the other hand not only receive concrete advice about how to organize a specific inspection and how to utilize PBR for their specific inspection implementation, but also how to evaluate software inspections in their projects and their organizations

References

Basili, V. R., 1997. Evolving and packaging reading technologies. *Journal of Systems and Software* 38(1).

Fagan, M. 1976, Design and code inspections to reduce errors in program development. *IBM Systems Journal* 15(3):182–211.

Gilb, T., and Graham, D. 1993. *Software Inspection*. Addison-Wesley Publishing Company.

Publications from the Thesis

Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sorumgard, S. and Zelkowitz, M. V. 1996. The empirical investigation of perspective-based reading. *Journal of Empirical Software Engineering* 2(1): 133–164.

Laitenberger, O., and Atkinson, C. 1999. Generalizing perspective-based inspection to handle object-oriented development artifacts. *Proceedings of the 21st International Conference of Software Engineering*.

Laitenberger, O., Atkinson, C., Schlich, M., and El Emam, K. 2000. An experimental comparison of reading techniques for defect detection in UML design documents. Accepted for publication in the *Journal of Systems and Software*, also published as ISERN-Technical Report 001.00/E.

Laitenberger, O., and DeBaud, J.-M. 2000. An encompassing life-cycle centric survey of software inspection. *Journal of Systems and Software* 1(50): 5–31.

Laitenberger, O., El Emam, K., and Harbich T. 2000. An internally replicated quasi-experimental comparison of checklist and perspective-based reading of code documents. Accepted for publication in *IEEE Transactions on Software Engineering*.

Oliver Laitenberger is currently a project/group leader at the the Fraunhofer Institute for Experimental Software

Engineering (IESE) in Kaiserslautern. His main interests are software quality assurance with software inspections, inspection measurement, and inspection improvement. As a researcher, Oliver Laitenberger has been working for several years in the development and evaluation of inspection technology. He has worked with several international companies in introducing and improving inspections. Oliver Laitenberger received the degree Diplom-Informatiker (M.S.) in computer science and economics from the University of Kaiserslautern, Germany, in 1996.