EVOLVING A PROCESS FOR INSPECTING OO DESIGNS

Guilherme H. Travassos[1†] travassos@cs.umd.edu

Forrest Shull[‡] fshull@fc-md.umd.edu

Jeffrey Carver[†] carver@cs.umd.edu

[†]Experimental Software Engineering Group
Department of Computer Science
University of Maryland at College Park
A.V. Williams Building
College Park, MD 20742
USA

[‡]Fraunhofer Center - Maryland
3115 Ag/Life Sciences Surge Bldg. (#296)
University of Maryland
College Park, MD 20742
USA

Resumo

A especificação de requisitos e projeto do software são atividades realizadas normalmente em momentos distintos utilizando níveis de abstração e perspectivas diferentes. Quando as atividades de projeto de alto nível terminam é possível inspecionar os documentos de projeto para verificar se são consistentes entre si e se conseguem representar corretamente e completamente os requisitos. Este artigo discute a definição e aplicação de técnicas de leitura (*Reading Techniques*) para inspeção de documentos de projeto orientado a objetos, apresentando suas definições, aplicação e, principalmente, de que forma estas técnicas vem sendo aprimoradas desde sua primeira versão através de ciclos de observação de uso, compreensão dos problemas e aprimoramentos dos procedimentos.

Abstract

Basically, requirements and design documents are built at different times, using a different viewpoint and abstraction level. When high-level design activities are finished, the documents can be inspected to verify whether they are consistent among themselves and if the requirements were correctly and completely captured. This paper discusses some issues regarding the definition and application of reading techniques to inspect high-level object-oriented design documents. It shows their definition, application and how such techniques have been evolving since their first version by iteration through a cycle of observing their use, understanding the problems, and improving the procedures.

Keywords: Reading Techniques, Software Inspection, Product Quality, Object-Oriented Software Quality, and Software Quality Evaluation

1. Introduction

A software inspection aims to guarantee that a particular software artifact is complete, consistent, unambiguous, and correct enough to effectively support further system development. For instance, inspections have been used to improve the quality of a system's design and code [1]. Typically, inspections require individuals to review a particular artifact, then to meet as a team to discuss and record defects, which are then sent to the document's author to be corrected. Most publications concerning software inspections have concentrated on improving the inspection meetings while assuming that individual reviewers are able to effectively detect defects in software documents on their own (e.g. [2,3]). However, empirical evidence has questioned the importance of team

[1] On leave from the Federal University of Rio de Janeiro – COPPE/Computer Science and System Engineering Department, partially supported by CAPES – Brazil.

This work was partially supported by UMIACS and by NSF grant CCR9706151.

In *Proc. of the Workshop on Software Quality, XIII Brazilian Symposium on Software Engineering*, Florianopolis, Curitiba, Brazil, October 1999.

meetings by showing that meetings do not contribute to finding a significant number of new defects that were not already found by individual reviewers [4,5].

“Software reading techniques” attempt to increase the effectiveness of inspections by providing procedural guidelines that can be used by individual reviewers to examine (or “read”) a given software artifact and identify defects. There is empirical evidence that software reading is a promising technique for increasing the effectiveness of inspections on different types of software artifacts, not just limited to source code [5,6,7,8,9,10]. Software reading can be performed on all documents associated with the software process, and is an especially useful method for detecting defects since it can be applied as soon as the documents are written. So, artifacts like requirements documents, use-cases, scenarios descriptions, design diagrams, source code and so on, could be read throughout the software lifecycle allowing developers to look for defects. Inspections are especially important in the case of design documents, since design defects (problems of correctness and completeness with respect to the requirements, internal consistency, or other quality attributes) can directly affect the quality of, and effort required for, the implementation of a system.

In this work, we discuss a set of reading techniques created specifically for inspections of high-level Object-Oriented (OO) designs. An OO design is a set of diagrams concerned with the representation of real world concepts as a collection of discrete objects that incorporate both data structure and behavior. Normally, high-level design activities start after the software product requirements are captured. So, concepts must be extracted from the requirements and described using the paradigm constructs. This means that requirements and design documents are built at different times, using a different viewpoint and abstraction level. When high-level design activities are finished, the documents (basically a set of well-related diagrams) can be inspected to verify whether they are consistent among themselves and if the requirements were correctly and completely captured. Table 1 presents our initial taxonomy of the types of defects that can be found in OO designs.

Type of Defect	***Description***
Omission	One or more design diagrams that should contain some concept from the general requirements or from the requirements document do not contain a representation for that concept.
Incorrect Fact	A design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
Inconsistency	A representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.
Ambiguity	A representation of a concept in the design is unclear, and could cause a user of the document (developer, low-level designer, etc.) to misinterpret or misunderstand the meaning of the concept.
Extraneous Information	The design includes information that, while perhaps true, does not apply to this domain and should not be included in the design.

Table 1 – Types of software defects, and their specific definitions for OO designs

This paper discusses some issues regarding the definition and application of reading techniques for high-level object-oriented design documents. Our research plan has been organized as follows:

1. Understand and model relevant aspects of the process for creating OO designs,
2. Define reading techniques that are tailored to OO design,
3. Run an experiment to investigate the feasibility of the reading techniques when used in the inspection of an actual OO design.

4. If the techniques are feasible, “fine tune”. Evolve the techniques by iterating through a cycle of observing their use, understanding the problems, and improving the procedures.
5. Use controlled, empirical studies to assess the effectiveness of the techniques versus other approaches to inspecting OO designs.

Our experiences in steps 1 through 3 are summarized elsewhere [11,12]. Section 3 of this paper summarizes our observations from the feasibility study (step 3). At this time we have reached step 4 of our research plan; sections 4 and 5 present our method of running observational studies and some results, and discuss the current version of the techniques. Section 6 summarizes this work and concludes with a discussion of issues still to be addressed.

2. An Initial Feasibility Study

Initially we performed a study to determine the *feasibility* of applying reading techniques to Object Oriented designs. This study was carried out in the context of an undergraduate software-engineering course at UMCP during the Fall 1998 semester. The course taught software-engineering principles, which students were required to apply to the development of an application over the course of the semester. The 44 students had a mix of previous experience: 32% had some previous industry experience in software design from requirements and/or use cases, 9% had no prior experience at all in software design, and the remaining majority of the class (59%) had classroom experience with design but had not used it on a real system. However, all students were trained in OO development, UML and OO software design activities as a part of the course. The students were organized in 15 teams (14 teams with 3 students each and 1 team with 2 students) to develop a small system, which contained some design complexity due to non-functional performance requirements.

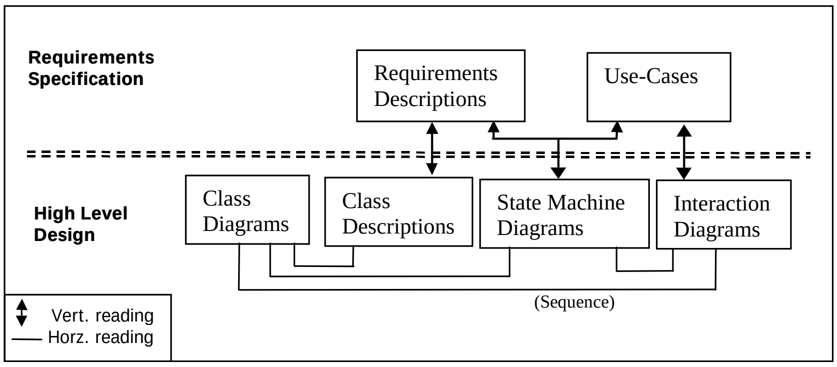

Figure 1 – Reading Techniques used in the experiment

As illustrated in Figure 1, we created two types of reading techniques, horizontal[2] and vertical[3]. The goal is that when all the techniques are used together, then all the documents in the design are covered. Each team applied the whole set of techniques, but responsibilities were divided in such a way that each team member applied only a small number of techniques. So, one subject dealt with the vertical reading, while the second and third divided the horizontal reading techniques between them. After individual review, the students met as teams in order to review their individual lists and to create a final list that reflected a group consensus of the defects in the document. During the process of performing the techniques, a number of metrics were collected so that the feasibility of the techniques could be evaluated. These metrics were both qualitative

[2] Horizontal reading refers to reading techniques that are used to read documents built in the same software lifecycle phase. Consistency among documents is the most important feature here.
[3] Vertical reading refers to reading techniques that are used to read documents built in different software lifecycle phases. Traceability between the phases is the most important feature here.

(e.g., how useful subjects thought the techniques were, how closely they had followed the procedure) and quantitative (e.g., the number and type of defects detected, time taken).

Results from the experiment showed some positive aspects as well as some areas for improvement for the next version of the techniques. Using the techniques did allow teams to detect defects (11 were reported, on average; 11.7 for horizontal readers, 10.4 for vertical), and in general students tended to agree that the techniques were helpful. Also, the vertical techniques tended to find more defects of omitted and incorrect functionality, while the horizontal techniques tended to find more defects of ambiguities and inconsistencies between design documents, lending some credence to the idea that the distinction between horizontal and vertical techniques is real and useful.

3. Lessons Learned from the Initial Study

However, qualitative information collected during the feasibility study also pointed to global aspects of the techniques that should be improved. The qualitative data were collected using questionnaires collected at the same time as the defect lists, retrospective questionnaires at the end of the semester, and post-hoc interviews (conducted separately for each team). Since the data was collected at multiple times, it was possible to check the consistency of subjects' answers over time, and to have more confidence in data accuracy. Our lessons learned from this qualitative data include the following:

Reading Technique for Class diagrams x Class descriptions

For each class modeled into the class diagram, do:

1) Read the **class description** to find an associated description to the class.
   ☞ Underline with a yellow pen the portion of the Class descriptions corresponding to the class

Verify if:

- **1.1) All the attributes are described and with basic types associated**
  ☞ Underline them with a blue pen
- **1.2) All the behaviors and conditions are described**
  ☞ Underline them with a green pen
- **1.3) All the class inheritance relationships are described**
  ☞ Draw a box around them with a yellow pen if there is any
- 1.4) All the class relationships (association, aggregation and composition) are described with multiplicity indication
  ☞ Circle each multiplicity indication with a blue pen if it is correct.

Figure 2 - First version of a horizontal reading technique. Note that the emphasis is on syntactic checking, that is, that the OO notation on both diagrams agrees.

Reading techniques should concentrate on semantic, not syntactic, issues: A majority of subjects in the feasibility study felt that reading techniques should explore more the semantic information involved in the models rather than concentrating on syntactic issues (Figure 2 illustrates the syntactic version). As a consequence, a new version of reading techniques was produced, which concentrated on semantic checking. By semantic checking we mean the validation of whether the design decisions made were reasonable and feasible. For example, a reader may be asked to check that appropriate types have been assigned to class attributes, given his or her understanding (from the requirements) of what information those attributes are supposed to represent. As another example, a reader may be given a procedure for how to check that some class (or combination of classes) is responsible for each functionality of the system, as described in the requirements. This new semantic version of the techniques required the reader to recreate the *rationales* that describe why a design appears the way it does, in response to the constraints and requirements set down in the requirements specification. This new version allowed the reading techniques to be less mechanical and require more thought on the part of the reader. As an example, Figure 3 highlights the new version for steps 1.1 and 1.2 of the reading technique previously shown in Figure 2.

Reading Technique for Class diagrams x Class descriptions

...

2) Read the **class descriptions** to find an associated description to each class on the class diagram.
Take a look at the class description. Can this description be used to describe the class that you are considering at this time? Is it using an adequate abstraction level?
☞ Box the class name in the class description with a blue pen and write in the same number given to the class on the class diagram.
☞ Mark found classes with a blue symbol (*) on the class diagram
Make good use of this time and verify if:

- **All the attributes are described along with basic types**
  Can this class encapsulate all these attributes? Are the attributes associated to feasible/possible basic types? Does it make sense to have these attributes in the class description? Is some attribute missing between the two documents or sounding extraneous?
  ☞ Write down missing and extraneous attributes with a yellow pen
- **All the behaviors and constraints are described**
  Can this class encapsulate all these behaviors? Is some behavior missing between the two documents or sounding extraneous? Are the behavior descriptions using the same abstraction level? Do the behaviors use the class attributes to accomplish the procedure? Are the constraints reachable using the attributes and behaviors of the class? Do the constraints make sense for this class? Are the constraints depending upon some class attributes? Were they described?
  ☞ Write down missing/extraneous behaviors with a green pen
  ☞ Write down missing/extraneous constraints with a yellow pen

...

Figure 3 - Second version of a horizontal reading technique. Note the introduction of semantic checking, that is, that the information expressed by the notation is *feasible, possible,* or *meaningful*.

Reading techniques need to include not only instructions for the reader, but some motivation as to why those instructions are necessary: Readers in the feasibility study also suggested that the reading technique should contain more information about *why* the different steps were necessary. When it was not clear why the readers needed to accomplish specific tasks, following the techniques could become an onerous and arbitrary undertaking rather than something perceived to be worthwhile. The mechanical nature of syntactic checking may also have contributed to this problem during the feasibility study.

The level of granularity of the instructions needs to be precisely described: Discussing functionality is a difficult but necessary part of the reading techniques. The difficulty comes from the many different levels of granularity at which system behavior can be described. To solve this problem we decided to define 3 distinct ways of discussing functionality, ranging from very specific system "behaviors" (the communications between objects that work together to implement system behavior) to system "services" (the steps by which some larger goal or functionality is achieved) and finally, at the top level, system functionalities (the behavior of the system, from the user's point of view). Thus, we can talk about specific behaviors of the classes combining to provide services of the system and, in turn, the high-level functionality that the user sees.

These were global changes to the techniques. That is, they affected every step of the procedures followed by the readers. However, we felt that to make sure the techniques could be practical, we needed feedback at another level of detail. Having already seen some evidence that using the techniques was feasible, we wanted to find specific improvements that could facilitate their use in practice. To do this we began to use observational research methods, in which a subject's application of the process is observed in some detail in order to gain insight into the process' use.

In this way, some individuals were asked to apply the techniques so that we could evaluate whether the identified problems had been addressed correctly. This feedback was important because it was possible to identify whether previous reader complaints had been addressed, or any new problems added. Using observational studies allows us to address our research goal of continuing to evolve the reading techniques in a more formal way, guided by actual experiences with their use, by identifying very specific areas requiring improvement.

4. Using Observational Studies to Evolve the Reading Process

Observational methods are distinct from *retrospective* methods of collecting qualitative data about processes, such as post-mortem interviews or questionnaires. We use the term "observational" to define a setting in which an experimental subject performs some task while being observed by an experimenter. The purpose of the observation is to collect data about how the particular task is accomplished. In our case, the subject has been given an OO reading technique and a set of artifacts. The observer is there to capture information about the circumstances in which the subject experiences problems or has trouble understanding the OO reading technique. The observer also takes note of the time consumed by each step of the process and whether or not the step was successful in finding defects. Although this observational technique can be a bit more time-consuming for the experimenter and not as relaxed for the subject as a simple post-mortem questionnaire, we think that it will deliver more accurate qualitative results.

Observational studies address an important drawback of retrospective techniques, namely that they do not present the experimenter with a completely accurate picture of the subject's thought process. This is because often, when using retrospective techniques, it is difficult for a subject to reconstruct from memory the actual thought process they went through to solve a problem. Another drawback to retrospective techniques is that if a subject is given time to reflect on what he did, then he may, intentionally or casually, present his thought process and ideas in a more structured and coherent way than they actually occurred. As our goal here is to design reading techniques that support the reader, knowledge of how the thought process actually works is important. Because of all this, it has been suggested that employing techniques to observe subjects while working may be a way to capture more complete and accurate information about what is really going on [13].

The technique of observing subjects performing different tasks in order to get some insight into the process has been used effectively in other areas of Software Engineering. The field of Human-Computer Interaction, in particular, is ripe with examples of these types of techniques. In his book on User Interface design, Shneiderman describes situations where users are observed while using a software interface. In this case, the users are asked to think aloud as they perform certain tasks using the interface. The experimenter uses this opportunity to listen for clues about the thought process the user employs while dealing with the interface, with hopes of finding ways to improve it [14]. Another example is discussed in [15], which describes the use of "think aloud" methods to gain some understanding of the thought process that software maintainers go through when trying to understand code. This is similar to our work, as we are trying to determine the thought process that a Object Oriented design reader will go through while trying to understand a particular design which is under review. Based on the success of observational techniques in these other areas of Software Engineering, we decided that using a similar approach could prove helpful in understanding how subjects use our OO reading techniques. We also hoped to gain insight into what parts of our OO reading techniques were helpful and what parts were confusing to the reader.

Data collection in an observational study can be split into 2 subtypes, observational and inquisitive. Observational data is collected while the process is being executed, but without direction from the researcher. For instance, subjects are told to think out loud as they execute a technique. This allows the researcher to gain insight into how the process is executed, for example, the researcher can record if the subject becomes confused or does not know what to do next at any step of the process. Collecting observational data is largely passive on the part of the researcher, since it is required that the researcher avoids interfering with the process being studied as much as possible [16].

Inquisitive data is collected at the completion of a process step, rather than during its execution. Data collection requires the researcher to be more assertive, since they must solicit responses to definite questions rather than observe process execution as it occurs. For example, at the end of each step, the researcher could ask the subject for qualitative feedback on the reading technique. This is not information that the subject would normally think about while executing the process, yet it is invaluable to collect at this time, while it is still fresh in the mind of the subject.

With these two types of questions in mind, we designed the guide that the researcher would follow when observing an experimental subject. The guide consisted of a copy of the reading technique (so that researchers could follow the subject's progress) in which each step was augmented with a series of questions to be filled in as the information became available. These questions asked for both observational and inquisitive data, and were decided upon after discussion among the research team of the most important aspects of the techniques to get feedback on.

5. Lessons Learned from Observational Studies

We have some experience at this point with observational studies on small numbers of subjects. The use of this type of study has resulted in the following lessons learned:

Time taken (observational data): It was observed that the time for applying such techniques is influenced by different aspects. The prior experience of the reader and the complexity of the information being inspected seem to be influential factors, although sometimes the order of steps and the abstraction level of the reading technique contributed to increase the time required for applying the techniques. For example, the first time that a particular reader used the techniques, extra time was required to understand the steps and even to understand the types of information in the inspected document that were relevant. But, after the reader applied the techniques once, the time spent to apply such techniques diminished. The observational techniques revealed several causes for this, including that the reader could better understand the steps and, interestingly, could mentally reorganize some of the steps in an order that made more sense for them.

It was also observed that the complexity of the class being inspected is an important factor that can determine how long particular steps will take. This aspect was noticed especially when readers tried to read classes belonging to inheritance hierarchies. In this situation, it was also observed that readers have a tendency to apply a top-down rather than a bottom-up approach. In other words, to understand the real-world concept or rationale behind the construction of the design, readers typically need to build the whole conceptual model before understanding the basic concept.

Problems encountered (observational data): It was possibly to identify three basic types of problems that were present in this version of the techniques: organizational issues, semantic issues, and the format of software artifacts being inspected.

Organizational problems are concerned with the way that the readers should apply the different reading techniques. Sometimes, readers suggested that applying one or another step before (or after) others could facilitate the identification of some features or information in the inspected document. For example, it was suggested that it is easier to understand the attributes of a class after the set of behaviors has been understood. This type of suggestion was possible only after readers had applied the techniques for the second or third time, having better understood them and having mentally reorganized the steps necessary to accomplish the tasks.

Another type of organizational problem regarded the format used to report defects. Readers uncovered several problems with using the defect taxonomy (Table 1) to report defects. Some confusion concerned the level of granularity that should be used. For instance, some students experienced difficulties trying to describe how a class description could be ambiguous when actually the ambiguity was related to only one attribute of the class.

Some semantic problems were also identified with the new reading technique version. Basically, these were concerned with how a reader's domain knowledge affects the execution of the techniques, and how questions should be phrased as part of the techniques to force readers to think about the semantic content of the information they were reading. Readers who used horizontal reading techniques reported that having some domain knowledge could speed up the reading process, allowing them to quickly recognize some concepts captured by the design diagrams. This situation was not detected in the vertical reading techniques, probably because readers were checking the design against the requirements description, which already aims to describe the necessary knowledge to understand the problem.

Last, but not least, some readers asked about the format of the artifacts being inspected. Although UML notation had been used during the construction process, specific changes to the format could be proposed to facilitate the reading process. As suggested by the readers, some organizational improvements (e.g. having an index to find the concepts quickly) could speed up searching consequently providing a way to speed up the reading process.

Influence of prior knowledge (inquisitive data): As mentioned before, some domain knowledge seemed to be necessary to support horizontal reading. Readers suggested that they didn't need to have the whole set of requirements to read design artifacts, but that some description of the system, its main concepts and objectives, would be worthwhile. It could be useful to justify the design choices or rationales somehow, with the goal of easing the identification of the concepts captured and used to represent the high-level system solution. Another point reported by the readers regarded the necessary level of object-oriented development expertise. Although the techniques were meant to be described to allow any reader to read design artifacts[4], readers seemed to be comfortable when they knew the basic concepts of the OO paradigm. For instance, the techniques provided examples and definitions described with enough detail to allow the reader to understand the idea behind each one of the constructs and concepts used. But, when different levels of abstraction were used (e.g., references to objects versus classes), readers without a more detailed knowledge regarding object-oriented paradigm encountered difficulty. It should be noted

[4] We assumed that readers could use the techniques without previous detailed knowledge about the OO constructs. This assumption is important mainly when vertical reading is been applied, because it allows the participation of readers who have knowledge of the system being constructed but not in-depth knowledge of software construction (e.g. system users or customers).

that this does not imply that readers need experience in OO development, just some knowledge of the basic OO concepts such as class, object, attribute, and behavior.

Another interesting comment is that readers typically knew of no other approach or tool that could elicit the types of information being checked by the reading techniques. The sole exception was for some of the steps of the horizontal reading techniques (mainly the ones dealing with syntactical issues), for which readers suggested using a CASE tool to automate the consistency check.

6. Open Questions and Conclusions

The object oriented reading techniques (OORTs) have been, and still are, evolving since their first definition. New issues and improvements have been included based on the feedback of readers and volunteers. Throughout this process, we have been trying to capture new features and to understand whether the latest reading technique version keeps its feasibility and interest. We have found observational techniques useful, because they allowed us to follow the reading process as it occurred, rather than trying to interpret the readers' post-hoc answers as was done before. Observing how readers normally try to read diagrams challenged many of our assumptions about how our techniques were actually being applied.

For now, a new and reorganized version of the OORTs, dealing with more semantic issues, has been created in an attempt to facilitate the reading process. These techniques are still organized into two distinct groups: horizontal techniques, which aim to be applied only among design artifacts, and vertical ones, which should be used when reading specification against design artifacts. It has been proposed through the feasibility study that horizontal reading seems to work as a consistency check, dealing more with syntactical rather than semantical issues, while vertical reading uses more meaningful information, acting as a traceability check. So, horizontal techniques are expected to be more mechanical and vertical techniques to require more thought on the part of the reader [Shull99].

However, two important questions remain open in this area. First, the role of domain knowledge is not yet well understood for these two sets of reading techniques, especially for horizontal reading. Since horizontal reading is a largely syntactic check of consistency between two design diagrams, it is not expected to require domain knowledge. Still, it has been observed that a reader possessing some knowledge about the problem domain seemed to be more effective than a reader who does not have the same level of knowledge. Some empirical investigation into exactly how domain knowledge plays a role in this type of reading could help us better understand and thus better support the process.

The second question regards the level of automated support that should be provided for such techniques. The observational studies have allowed us to understand which steps of the techniques can feel especially repetitive and mechanical to the reader. So, the clerical activities regarding the reading process using OORTs must be precisely defined and identified. For this situation, further observational studies play an important role and they should be executed aiming to collect suggestions on how to automate the clerical activities concerned with OORTs.

Acknowledgements

The authors would like to thank Dr. Mikael Lindvall (Fraunhofer Center – Maryland) whose suggestions collaborated to improve the reading techniques. We also recognize the support, management and dedication of Prof. Victor R. Basili for this research work.

References

[1] M. E. Fagan. “Design and Code Inspections to Reduce Errors in Program Development.” *IBM Systems Journal*, 15(3):182-211, 1976.

[2] M. Fagan; “Advances in Software Inspections.” *IEEE Transactions on Software Engineering*, 12(7): 744-751, July 1986.

[3] T. Gilb, D. Graham. *Software Inspection*. Addison-Wesley, reading, MA, 1993.

[4] L. G. Votta Jr. “Does Every Inspection Need a Meeting?” *ACM SIGSOFT Software Engineering Notes*, 18(5): 107-114, December 1993.

[5] A. Porter, L. Votta Jr., V. Basili. “Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment.” *IEEE Transactions on Software Engineering*, 21(6): 563-575, June 1995.

[6] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, M. V. Zelkowitz. “The Empirical Investigation of Perspective-Based Reading.” *Empirical Software Engineering Journal*, I, 133-164, 1996

[7] V. Basili, G. Caldiera, F. Lanubile, and F. Shull. “Studies on Reading Techniques.” In *Proc. of the Twenty-First Annual Software Engineering Workshop*, SEL-96-002, pages 59-65, Greenbelt, MD, December 1996.

[8] P. Fusaro, F. Lanubile, and G. Visaggio. “A Replicated Experiment to Assess Requirements Inspections Techniques.” *Empirical Software Engineering Journal*, vol.2, no.1, pp.39-57, 1997.

[9] F. Shull. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis, University of Maryland, College Park, December 1998.

[10] Z. Zhang, V. Basili, and B. Shneiderman, “An Empirical Study of Perspective-Based Usability Inspection.” Human Factors and Ergonomics Society Annual Meeting, Chicago, Oct. 1998.

[11] F. Shull, G. Travassos, V. Basili. “Towards Techniques for Improved OO Design Inspections.” Accepted at the Workshop on Quantitative Approaches in Object-Oriented Software Engineering (in association with the 13th European Conf. on Object-Oriented Programming), Lisbon, Portugal, 1999.

[12] G. Travassos, F. Shull, M. Fredericks, V. Basili. “Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Improve Software Quality.” Accepted at the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Denver, Colorado, 1999.

[13] M.W. Van Someren, Y.F. Barnard, and J.A.C Sandberg. *The Think Aloud Method: A Practical guide to Modeling Cognitive Processes.* Academic Press: London. 1994.

[14] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley. 1998

[15] A. von Mayrhauser and A.M. Vans. “Industrial experience with an integrated code comprehension model.” *Software Engineering Journal*. September 1995. p 171-182.

[16] J. Singer and T. Lethbridge. “Methods for Studying Maintenance Activities.” In *Proc. of the Workshop for Empirical Studies of Software Maintenance*, 1996