Reading Techniques for OO Design Inspections

Guilherme H. Travassos[†,•] travassos@cs.umd.edu
Forrest Shull[‡] fshull@fraunhofer.org
Jeffrey Carver[†] carver@cs.umd.edu
Victor R. Basili[†,‡] basili@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

[•]**Computer Science and System Engineering Department COPPE**
Federal University of Rio de Janeiro
C.P. 68511 - Ilha do Fundão
Rio de Janeiro – RJ – 21945-180
Brazil

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

ABSTRACT

Inspections can be used to identify defects in software artifacts. In this way, inspection methods help to improve software quality, especially when used early in software development. Inspections of software design may be especially crucial 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. We have created a set of "reading techniques" (so called because they help a reviewer to "read" a design artifact for the purpose of finding relevant information) that gives specific and practical guidance for identifying defects in Object-Oriented designs. Each reading technique in the family focuses the reviewer on some aspect of the design, with the goal that an inspection team applying the entire family should achieve a high degree of coverage of the design defects.

In this paper, we present an overview of this new set of reading techniques. We discuss the reading process and how readers can use these techniques to detect defects in high level object oriented design UML diagrams.

Keywords: OO Design, Reading Techniques, Software Quality, and Software Inspection

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 [Fagan76]. Typically, inspections require individuals to review a particular artifact, then 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. [Fagan86, Gilb93]). However, empirical evidence has questioned the importance of team meetings by showing that meetings do not contribute to finding a significant number of new defects that were not already found by individual reviewers [Votta93, Porter95].

"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. These techniques consist of a concrete **procedure** given to a reader on what information in the document to look for. Another important component of the techniques are the **questions** that explicitly ask the reader to think about the information just uncovered in order to find defects. In previous work, we have developed families of reading techniques [Basili96]. 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 [Porter95, Basili96, Basili96b, Fusaro97, Shull98, Zhang98]. In this work, we concentrate specifically on inspections, for the purpose of defect detection, of high-level Object-Oriented (OO) designs diagrams represented using UML [Fowller97]. (UML is a notational approach that does not define how to organize development tasks.) Figure 1 organizes the "problem space" to which reading techniques can be applied, and illustrates how reading techniques for this task (known as Traceability-Based Reading) fit with previous work. Families of reading techniques have been tailored to defect inspections of requirements (for requirements expressed in English or SCR, a formal notation) and to usability inspections of user interfaces.

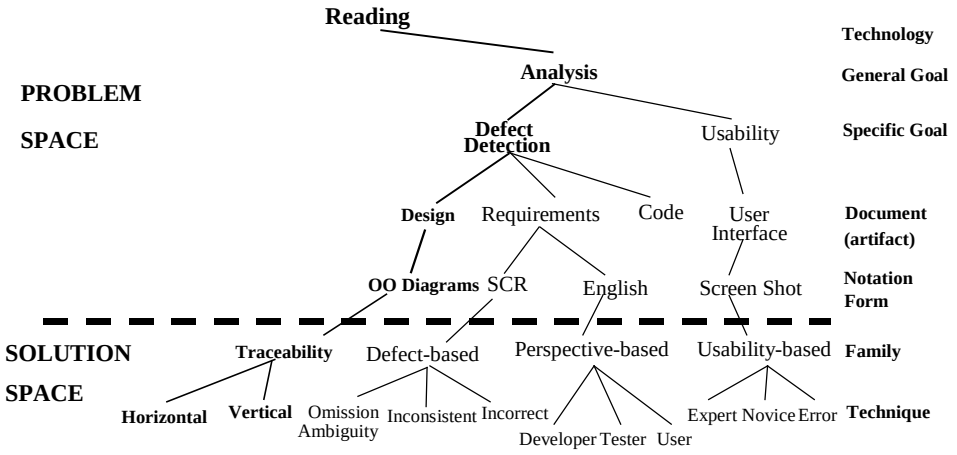

Figure 1 –Families of OO Reading Techniques

Section 2 briefly describes object oriented design in terms of the information that is important to be checked during software inspections. Section 3 introduces the reading techniques, showing the different types of defects such techniques are intended to identify and an outline of the whole set of techniques. The fourth section discusses how the techniques can be used for inspecting OO designs. Finally, some suggestions for future work are discussed in the conclusions.

2. Object Oriented Designs in UML

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. High-level design activities deal with the problem description but do not consider the constraints regarding it. That is, these activities are concerned with taking the functional requirements and mapping them to a new notation or form, using the paradigm constructs to represent the system via design diagrams instead of just a textual description. Such an approach allows developers to understand the problem rather than to try to solve it.

Low-level design activities deal with the possible solutions for the problem; they depend on the results from the high-level activities and nonfunctional requirements, and they serve as a model for the code. Our interest is to define reading techniques that could be applied on high-level design documents. We feel that reviews of high-level designs may be especially valuable since they help to ensure that developers have adequately understood the problem before defining the solution. Since low-level designs use the same basic diagram set as the high-level design, but using more detail, reviews of this kind can help ensure that low-level design starts from a high-quality base.

More specifically, the reading techniques investigated in this work are tailored to inspections of documents using UML notation. UML diagrams capture the static and dynamic view of the real world as described by the object-oriented constructs. We focused our reading techniques on the following high-level design diagrams: class, interaction (sequence and collaboration), state machine and package. Usually, these are the main UML diagrams that developers build for high-level OO design. They capture the static and dynamic views of the problem, and even allow the teamwork to be organized, based on packaging information. The design content needs to be compared against the requirements, which can likewise be described using a number of separate diagrams to capture different aspects. In particular, we expect that there will be a textual description of the functional requirements that may also describe certain behaviors using more specialized representations such as use-cases [Jacobson95].

Thus, we identify the following as important sources of information for ensuring the quality of a UML high level design:

- A set of functional requirements that describes the concepts and services that are necessary in the final system;
- Use cases that describe important concepts of the system (which may eventually be represented as objects, classes, or attributes) and the services it provides;
- A class diagram (possibly divided into packages) that describes the classes of a system and how they are associated;
- A set of class descriptions that lists the classes of a system along with their attributes and behaviors;
- Sequence diagrams that describe the classes, objects, and possibly actors of a system and how they collaborate to capture services of the system;
- State diagrams that describe the internal states in which a particular object may exist, and the possible transitions between those states.

3. Reading Techniques for high-level design

Each reading technique can be thought of as a set of procedural guidelines that reviewers can follow, step-by-step, to examine a set of diagrams and detect defects. The types of defects on which our techniques are focused, as listed in Table 1, are based on earlier work with requirements inspections. The defect taxonomy is important since it helps focus the kinds of questions reviewers should answer during an inspection.

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

We defined one reading technique for each pair or group of diagrams that could usefully be compared against each other. For example, use cases needed to be compared to interaction diagrams to detect whether the functionality described by the use case was captured and all the concepts and expected behaviors regarding this functionality were represented. The full set of our reading techniques is defined as illustrated in Figure 2, which differentiates horizontal[1] (comparisons of documents within a single lifecycle phase) from vertical[2] (comparisons of documents between phases) reading.

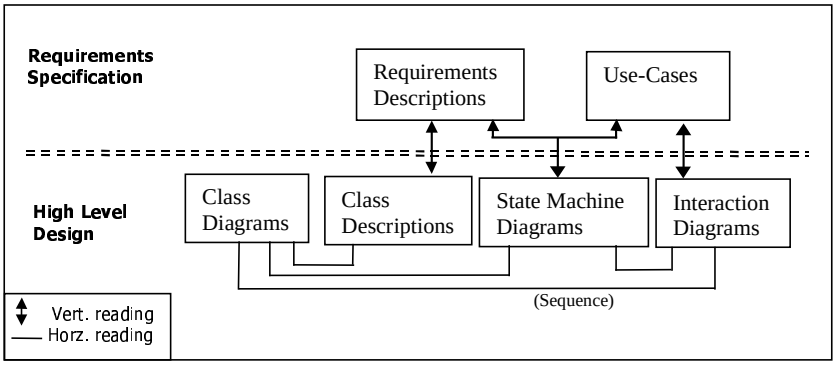

Figure 2 – Set of OO Reading Techniques

[1] Consistency among documents is the most important feature here.
[2] Traceability between the phases is the most important feature here.

Initial validation of these techniques was accomplished by means of a study [Shull99, Travassos99] that provided evidence for the feasibility of these techniques. Using the techniques did allow teams to detect defects, and in general subjects agreed 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 [Travassos99].

Further studies have been undertaken to improve the practical applicability of the techniques. As a result of specific feedback from the feasibility study, we developed a second version of the techniques and studied them using an observational approach (i.e., using experimental methods suitable for understanding the process by which subjects apply the techniques) [Travassos99b]. The feasibility study had identified *global* issues for improvement, that is, issues that affected the entire process, such as the amount of semantic versus syntactic checking. The observational approach was necessary to understand what improvements might be necessary at the level of individual steps, for example, whether subjects experience difficulties or misunderstandings while applying the technique (and how these problems may be corrected), whether each step of the technique contributes to achieving the overall goal, and whether the steps of the technique should be reordered to better correspond to subjects' own working styles. Detailed information about the results and also an improved version of the techniques can be found in [Shull99b].

4. Using OO Reading Techniques for inspecting OO Design

In this section we explore the application of the reading techniques in an inspection process. While horizontal reading aims to identify whether all of the design artifacts are describing the same system, vertical reading tries to verify whether those design artifacts represent the right system, which is described by the requirements and use-cases. So, the goal is that when all the techniques are used together, then all the quality issues in the design are covered. The development team can use the whole set of the techniques, but if some design artifacts do not exist, there is no impact on the design inspection process. A subset or reordering of the techniques may also be chosen based on important attributes of the design to be reviewed. This is particularly interesting when developers are dealing with specialized application domains. For example, consider a system whose functionality is based mainly on its reaction to stimuli where state machine diagrams are common. In this situation, it could be beneficial to use the reading techniques that focus on state machine diagrams before using the reading techniques that focus on the other design diagrams. For conventional systems, such as database systems, the semantic model of the information and the flow of the transactions seem to be the important information. Therefore, a subset of the techniques could be picked that focus on this information. In this situation, first reading the class diagram against the sequence diagrams seems to be a good idea then continuing with the rest of the techniques.

To organize the reading process, reading responsibilities can de distributed among the members of the inspection team, reducing the reading effort per team member and improving the reading process. In this way, each one of the readers can apply a reduced number of reading techniques, or even deal with a reduced number of artifacts at the same time. After individual review, it is important to organize a meeting in order to review each one of the individual defect lists and to create a final list that reflected a group consensus of the defects in the documents. It is not necessary to apply the techniques in a particular order, but it seems to be reasonable to apply first horizontal reading for all existing design artifacts and then vertical reading, to ensure that a consistent system description is checked against the requirements. In Figure 3 is an example of how the techniques could be organized among a team of three reviewers.

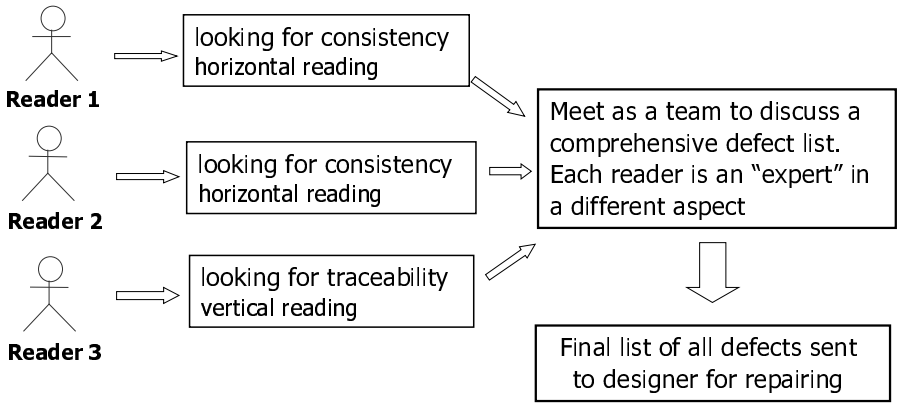

Figure 3 – Organizing reading with 3 readers

To support these two types of reading (horizontal and vertical) we have introduced some new terminology to describe the actions of the system. First, because the level of abstraction and granularity of the information in the requirements and use-cases is different from the abstraction and information in the design artifacts, the concept of *system functionality* was broken down into three complementary concepts (messages, services, and functionality). Messages are the very lowest-level behaviors out of which system services and, in turn, functionalities are composed. They represent the communication between objects that work together to implement system behavior. Messages may be shown on sequence diagrams and must be associated with class behaviors. Services are combinations of one or more messages and usually capture some basic activity necessary to accomplish a functionality. They can be considered low-level actions performed by the system. They are the "atomic units" out of which system functionalities are composed. A service could be used as a part of one or more functionalities. We use the term "functionality" to describe the behavior of the system from the user's point of view, in other words, the functionality that the user expects to be visible. A functionality is composed of one or more services. Users do not typically consider services an end in themselves; rather, services are the steps by which some larger goal or functionality is achieved.

A second important piece of terminology is that of conditions and constraints. A condition describes what must be true for the functionality to be executed. A constraint must always be true for system functionality. This information is important to readers comparing different diagrams

since it describes *how* the functionality must be implemented; this information is important to maintain with the functionality it describes.

B.2 Reading 2 -- State diagrams x Class description

Goal: To verify that the classes are defined in a way that can capture the functionality specified by the state diagrams.

Inputs to Process: A set of class descriptions that lists the classes of a system along with their attributes and behaviors and a state diagram that describes the internal states in which an object may exist, and the possible transitions between states.

For **each state diagram,** perform the following steps:

1) **Read the state diagram to understand the possible states of the object and the actions that trigger transitions between them.**
2) **Find the class or class hierarchy, attributes, and behaviors on the class description that correspond to the concepts on the state diagram.**
3) **Compare the class diagram to the state diagram to make sure that the class, as described, can capture the appropriate functionality.**

   Using your semantic knowledge of this class and the behaviors it should encapsulate, are all states described? If not, you have uncovered a defect of incorrect fact, that is, the class as described cannot behave as it should.

   Is there some unstarred state? Could you evaluate the importance of this state? Does it really describe an essential object state? Is the state feasible considering all actions and constraints surrounding it? If yes, probably something is missing on the class diagram and there is an inconsistency between the diagrams. Otherwise, an extraneous fact should be reported.

   Is there some unstarred event? If yes, fill in a defect record showing the inconsistency between the class description and state diagram.

   Is there some unstarred constraint? Is the constraint directly concerned with some object data? If yes, fill in a defect record showing the information that has been omitted from the class description.

Figure 4 – An excerpt of a Horizontal Reading

The main idea in applying horizontal reading is to understand whether all the high level design artifacts are representing the same system. We must keep in mind that the artifacts should model the same system information but from different perspectives. UML organizes the artifacts and different types of information based on the type of system information they contain. There are specific artifacts to capture essentially static information (basically, the structure assumed by the domain's objects while playing specific roles in the problem domain) and specific artifacts to capture essentially dynamic information (basically, the consequences when objects are asked to behave in order to accomplish system functionalities). These different views are useful and together allow developers to understand what is going on with the objects and how they are accomplishing the required functionalities in the context of the problem. However, these differences among the diagrams make the inspection process a bit more complicated. For instance, when comparing sequence diagrams against state machine diagrams two different perspectives must be combined to interpret and identify possible defects. Each one of the sequence diagrams is a represents some system objects and the messages exchanged between them that implement some functionality required by the user while, on the other hand, the state machine diagram is a picture of what happens to one object when it is influenced by the events occurring in multiple sequence diagrams. Sequence diagrams show the specific messages exchanged by objects, while state diagrams show how the system responds to events, which can be messages, services, or functionality. Both diagrams must convey information about conditions

and constraints on the functionality. So, the horizontal reading techniques explore these types of differences and help reduce the semantic gap between the documents. Figure 4 shows an excerpt from a horizontal reading technique highlighting the concerns for each one of the reading steps (some details are omitted).

B.7 Reading 7 -- State Diagrams x Requirements Description and Use-cases
Goal: To verify that the state diagrams describe appropriate states of objects and events that trigger state changes as described by the requirements and use cases.
Inputs to process: The set of all state diagrams, each of which describes an object in the system. A set of functional requirements that describes the concepts and services that are necessary in the final system and the set of use cases that describe the important concepts of the system

For each state diagram, do the following steps:

1) **Read the state diagram to basically understand the object it is modeling.**
2) **Read the requirements description to determine the possible states of the object, which states are adjacent to each other, and events that cause the state changes.**
3) **Read the Use cases and determine the events that can cause state changes.**
4) **Read the state diagram to determine if the states described are consistent with the requirements and if the transitions are consistent with the requirements and use cases.**

Were you able to find all of the states?

If a state is missing, look to see if two or more states that you marked in the requirements were combined into one state on the state diagram. If not, then you have found a defect of Omission. If so, then does this combination make sense? If not, you have found a defect of Incorrect Fact.

Were there extra states in the state diagram?

Look to see if one state that you marked in the requirements has been split into two or more states in the state diagram. If not, then you have found a defect of Extraneous. If so, does this split make sense? If not, you have found a defect of Incorrect Fact.

Do all of the events on the adjacency matrix appear on the state diagram? If not, you have found a defect of omission. Do events appear on the state diagram that are not on the adjacency matrix? If so, you have found a defect of extraneous fact.

Did you find all of the constraints that are on the adjacency matrix? If not, then you have found a defect of omission. Did you find a constraint on the state diagram that is not on the adjacency matrix? If so, does the constraint make sense? If not then you have found a defect of extraneous fact.

Figure 5 – An excerpt of a Vertical Reading

To apply vertical reading readers should be aware of the differences between the two lifecycle phases in which the documents were created and how the traceability between these two different phases could be explored. The levels of abstraction and information representation between these phases are quite different. Requirements and use cases should precisely describe the problem and thus use a totally different representation than the design artifacts. Moreover, usually the entire problem definition is presented using these two types of document. There is no separation of concerns and no direct mapping from one phase (specification) to another (design). Vertical reading techniques explore such ideas and provide some guidance to help the reader identify the information s/he needs. For example, the requirements descriptions and use cases capture the functionality of the entire system and in some cases the services, but not the messages. Designers using these requirements and use cases decide about the messages based on the viewpoint (abstraction) used to classify and organize the classes. Sequence diagrams are organized based on messages that work together in some way to provide the services, which compose the required functionality. Requirements and use cases describe constraints and conditions in general terms;

on a sequence diagram such information must be made explicit and associated with the appropriate messages. So, vertical reading techniques explore these types of differences by defining some guidelines for tracing the right information between these two lifecycle phases. Figure 5 shows an excerpt from a vertical reading technique highlighting the concerns for each one of the reading steps (some details are omitted).

A full description of the entire set of techniques, including the ones referred to here, can be found in [Shull99b], which is accessible via the web.

5. Ongoing Work

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 version of the reading techniques keeps its feasibility and interest. We have found observational techniques useful, because they have allowed us to follow the reading process as it occurred, rather than trying to interpret the readers' post-hoc answers as we have done in the past. Observing how readers normally try to read diagrams challenged many of our assumptions about how our techniques were actually being applied.

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.

Currently, the techniques are undergoing experimental evaluation, which is aimed at evolving them. In each experiment we explore a different issue regarding the techniques in order to evolve them or understand them at a deeper level. This series of experiments is an evolutionary process. The feedback from the readers and the observation of the techniques usage are playing an important role as we work towards a useful and feasible set of reading techniques for OO design. The results of these experiments will be published in future publications, which will be available at http://www.cs.umd.edu/projects/SoftEng/ESEG.

Acknowledgements

This work was partially supported by UMIACS and by NSF grant CCR9706151.
Dr. Travassos also recognizes the partial support from CAPES- Brazil.

References

[Basili96] 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.

[Basili96b] 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.

[Fagan76] M. E Fagan. "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal, 15(3):182-211, 1976.

[Fagan86] M. Fagan. "Advances in Software Inspections." IEEE Transactions on Software Engineering, 12(7): 744-751, July 1986.

[Fowller97] M. Fowller, K. Scott. UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, 1997.

[Fusaro97] 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.

[Gilb93] T. Gilb, D. Graham. Software Inspection. Addison-Wesley, reading,MA, 1993.

[Jacobson95] I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard. Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, revised printing, 1995.

[Porter95] 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.

[Shull98] F. Shull. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis, University of Maryland, College Park, December 1998.

[Shull99] F. Shull, G. Travassos, V. Basili. Towards Techniques for Improved OO Design Inspections. Workshop on Quantitative Approaches in Object-Oriented Software Engineering (in association with the 13th European Conf. on Object-Oriented Programming), Lisbon, Portugal, 1999. On line at http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/postscript/ecoop99.ps.

[Shull99b] Forrest Shull, Guilherme H. Travassos, Jeffrey Carver, Victor R. Basili. Evolving a Set of Techniques for OO Inspections. Technical Report CS-TR-4070, UMIACS-TR-99-63, University of Maryland, October 1999. http://www.cs.umd.edu/Dienst/UI/2.0/Describe/ncstrl.umcp/CS-TR-4070

[Travassos99] G. Travassos, F. Shull, M. Fredericks, V. Basili. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Improve Software Quality. In the Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Denver, Colorado, 1999.

[Travassos99b] Guilherme H. Travassos, Forrest Shull, Jeffrey Carver. Evolving a Process for Inspecting OO Designs. XIII Brazilian Symposium on Software Engineering: *Workshop on Software Quality*. Florianópolis, Curitiba, Brazil, October 1999. On line at http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/ postscript/wqs99.ps.

[Votta93] L. G.Votta Jr. "Does Every Inspection Need a Meeting?" ACM SIGSOFT Software Engineering Notes, 18(5): 107-114, December 1993.

[Zhang98] 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.