Studies on Reading Techniques

Victor Basili, Gianluigi Caldiera, Filippo Lanubile, and Forrest Shull

Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland, MD, USA
{basili | gcaldiera | lanubile | fshull}@cs.umd.edu

1. Introduction

Software reading is a key technical activity that aims at achieving whatever degree of understanding is needed to accomplish a particular objective. The various work documents associated with software development (e.g., requirements, design, code, and test plans) often require continual understanding, review and modification throughout the development life cycle. Thus software reading, i.e., the individual analysis of textual software work products, is the core activity in many software engineering tasks: verification and validation, maintenance, evolution, and reuse.

Through our work in the SEL, we have evolved our understanding of reading technologies via experimentation. We have run empirical studies ranging from blocked subject-project experiments (reading by step-wise abstraction vs. functional and structural testing [Basili,Selby87]) to replicated projects (University of Maryland Cleanroom study [Selby,Basili,Baker87]) to a case study (the first SEL Cleanroom study) to multi-project variation (the set of SEL Cleanroom projects [Basili,Green94]) and most recently, back to blocked subject-project experiments (scenario-based reading vs. current reading [Basili,Green, Laitenberger,Lanubile,Shull,Soerumgaard,Zelkowitz96], [Porter,Votta,Basili95]).

We have used a variety of experimental designs to provide insight into the effects of different variables on reading. The experiments are based upon the ideas that reading is a key technical activity for improving the analysis of all kinds of software documents and that we need to better understand its effect. We believe these studies demonstrate the evolution of knowledge about reading, experimentation, and the packaging of experimental results over time. Several of these experiments have been replicated by other researchers.

To provide a technological base to software reading, we attempt to develop specific reading techniques, made up of a concrete set of instructions which are given to the reader on how to read or what to look for in a software work product. Our current research efforts focus on the development of families of reading techniques, based on empirical evaluation. Each family of reading techniques can be parameterized for use in different contexts and must be evaluated for those contexts.

The taxonomy of reading families is shown in Figure 1. The upper part of the tree (over the dashed horizontal line) models the problems that can be addressed by reading. Each level represents a further specialization of the problem according to classification attributes which are shown in the rightmost column of the figure. For example, reading (*technology*) can be applied for analysis (*high level goal*), more specifically to detect faults (*specific goal*) in a requirements specification (*document*) which are written in English (*notation/form*).

The lower part of the tree, (below the dashed horizontal line) models the specific solutions we have provided to date for the particular problems, represented by each path down the tree. The solution space consists of reading families and reading techniques. Each family is associated with a particular goal, document or software artifact, and notation in which the document is written. Each technique within the family is: (1) tailorable, based upon the project and environment characteristics; (2) detailed, in that it provides the reader with a well-defined set of steps to follow; (3) specific, in that the reader has a particular purpose or goal for reading the document and the procedures that support the goal; (4) focused, in that it provides a particular coverage of the document, and a combination of techniques in the family provides coverage of the entire document. Finally each technique is being studied empirically to determine if and when it is most effective.

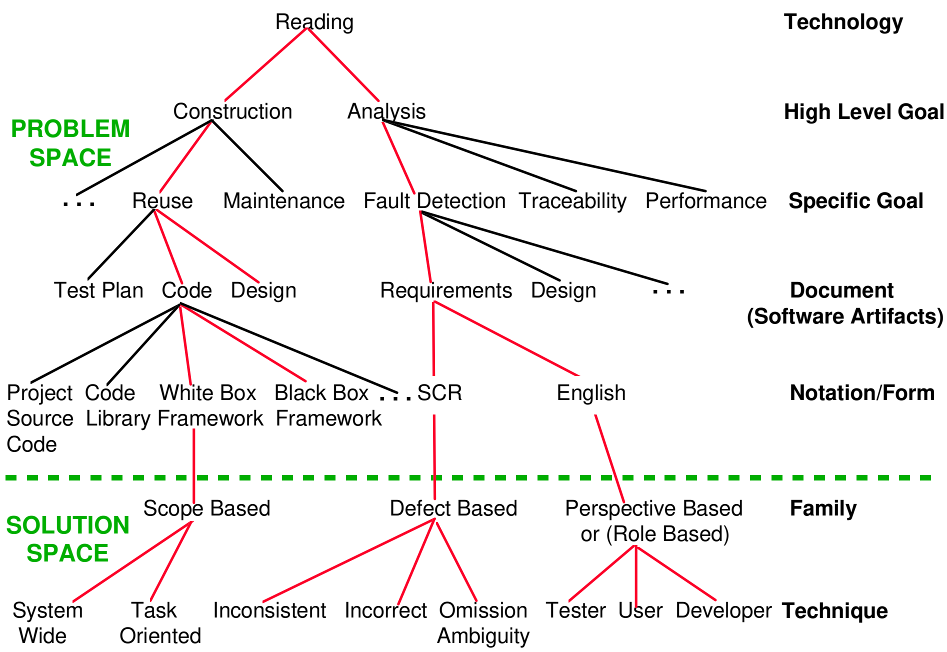

Figure 1. Families of reading techniques

Each software life cycle phase contains both construction and analysis activities. The design phase, for example, is responsible for creating design documents, as well as for analyzing their quality. Since construction and analysis are two parts of the same phase, you can learn from analysis technologies about construction technologies. At a high level, we divide reading activities into Reading for Analysis and Reading for Construction, to parallel this distinction between analysis and construction processes and to show that the usefulness of good reading techniques is not limited to any narrow portion of the software life-cycle. The next two sections describe our work in these areas.

2. Reading for Analysis

Reading for analysis is aimed at answering the following question: Given a document, how do I assess various qualities and characteristics? Reading for analysis is important for product quality; it can help us understand the types of defects we make, and the nature and structure of the product. It can be used for various documents throughout the life cycle. Besides helping us assess quality, it can provide insights into better development techniques.

Our research into reading for analysis has so far emphasized *defect detection*; we have focused on the requirements phase for this purpose. We have generated two families of reading techniques (collectively known as *scenario-based reading*), by creating operational scenarios which require the reader to first create an abstraction of the product, and then answer questions based on analyzing the abstraction with a particular emphasis or role that the reader assumes. Each reading technique in a family can be based upon a different abstraction and question set. The choice of abstraction and the types of questions may depend on the document being read, the problem history of the organization, or the goals of the organization.

The first family of scenario-based reading techniques is known as *defect-based reading*, and focuses on a model of the requirements using a state machine notation. The different model views are based upon focusing on specific defect classes: data type inconsistency, incorrect functions, and ambiguity or missing information. The analysis questions are generated by combining and abstracting a set of questions that are used in checklists for evaluating the correctness and reliability of requirements documents.

The second family of techniques, *perspective-based reading*, focuses on different product perspectives, e.g., reading from the perspective of the software designer, the tester, the end-user, the maintainer, the hardware engineer. The analysis questions are generated by focusing predominantly on various types of requirements errors (e.g., incorrect fact, omission, ambiguity, and inconsistency) by developing questions that can be used to discover those errors from the one perspective assumed by the reader of the document (e.g., the questions for the tester perspective lead the reader to discover those requirement errors that could be found by testing the final product).

In order to understand the effectiveness of scenario-based reading techniques in particular, we have experimentally studied techniques from both families. The first series of experiments [Porter,Votta,Basili95], [Basili, Green, Laitenberger, Lanubile, Shull, Soerumgaard, Zelkowitz96] was aimed at discovering if scenario-based reading is more effective than current practices. Based upon these experiments, we had empirical evidence that scenario-based reading techniques can improve the effectiveness of reading methods. At the same time, we noted that some scenarios were less effective than others. We give some details of these experiments here in order to illustrate our own experiences with experimentation in software engineering.

2.1 Defect-Based Reading Experiment

In the defect-based reading study [Porter,Votta,Basili95], we evaluated and compared defect-based reading, ad hoc reading and checklist-based reading, with respect to their effect on fault detection effectiveness in the context of an inspection team. The study, a blocked subject-project, was replicated twice in the spring and fall of '93 using 48 graduate students at the University of Maryland. The experimental design was a partial fractional factorial design. The design was less elegant than the [Basili,Selby87] design because the comparison here is with the status quo approach (ad hoc) or with a less procedurally organized approach (checklists) so it is impossible to teach the subject a defect-based reading approach and then return to ad hoc or check list. In this case, a sort of ordering was assumed. On the first pass there were more ad hoc and check list readers. Several, but not all, were moved to defect-based reading on the second pass.

Major results were that:

- the defect-based readers performed significantly better than ad hoc and checklist readers;
- the defect-based reading procedures helped reviewers focus on specific fault classes but were no less effective at detecting other faults; and
- checklist reading was no more effective than ad hoc reading.

2.2 Perspective-Based Reading Experiment

In the perspective-based reading study [Basili, Green, Laitenberger, Lanubile, Shull, Soerumgaard, Zelkowitz96], we evaluated and compared perspective-based reading and NASA's current reading technique with respect to their effect on fault detection effectiveness in the context of an inspection team. Three types of perspective-based reading techniques were defined and studied: tester-based, designer-based, and user-based. The study, again a blocked subject-project, was run twice in the SEL environment with NASA professionals.

The design evaluated the effectiveness of perspective-based reading on both domain-specific and generic requirements documents, which had been constructed expressly so that the generic portion could be replicated in a number of different environments, while the domain-specific part could be replaced in each new environment. This would allow us to combine the generic parts from multiple studies but focus on improvement local to a particular environment. Based on feedback from the subjects and other difficulties encountered in the first run of the experiment, we were able to make changes to the experimental design that improved the second run. For example, we found it necessary to:

- Include more training sessions, to make certain that subjects were familiar with both the documents and techniques involved in the experiment;

- Allow less time for each review of the document, since subjects tended to tire in longer sessions;
- Shorten some of the documents, to reach a size that could realistically be expected to be checked in an experimental, time-constrained setting.

Major results of this experiment were that:

- both team and individual scores improved when perspective-based reading was applied to generic documents
- team scores improved when perspective-based reading was applied to NASA documents

Although the true benefit of PBR is expected to be seen at the level of teams which combine several different perspectives for improved coverage, the results for individuals showed that the use of PBR may lead to improvements at the individual level as well. Thus, we further analyzed the individual reviewers' performance with the generic documents considering other attributes of effectiveness. Preliminary results of this second-round analysis were that:

- PBR reviewers took more time than reviewers using their current reading technique but the average cost for finding a defect was the same for both the methods
- The percentage of false positives for both methods is about the same. There were less false positives with PBR although the difference was not significant)

If we consider that PBR reviewers found more defects than reviewers using their current reading technique and assume that the cost of finding a defect increases as more defects are found, we can conclude (for generic documents) that:

- PBR is actually more productive than the local reading technique.
- The relative effort spent fixing defects is better for PBR.

By tailoring the perspectives also to the NASA application domain, we should be able to improve individual performance on these tasks. We need to improve the treatments used in the reading techniques. This can be done by developing questions for each scenario using the specific application domain (e.g., flight dynamics requirements documents), by focusing on the abstraction mechanism used (e.g., using a specific technique like equivalence partition testing for the testing perspective), or focusing the questions to cover certain classes of defects more effectively.

We need to add a qualitative component to the controlled studies to gather more insights into what is needed to better set up the experiment, define the terminology, and interpret the results. For example, controlled experiments could be supplemented with various standard methods in qualitative analysis such as the use of pre-tests, post-tests, ethnographic studies, and interviews.

3. Reading for Construction

Reading for construction is aimed at answering the question: Given an existing system, how do I understand how to use it as part of my new system? Reading for construction is important for comprehending what a system does, what capabilities exist and do not exist; it helps us abstract the important information in the system. It is useful for maintenance as well as for building new systems from reusable components and architectures.

Our emphasis here has so far focused on the *reuse* of an existing system or library. Reusing class libraries does increase quality and productivity, but class libraries do not provide default system behavior but only functionality at a low level, and force the developer to provide the interconnections between the libraries. Greater benefits can be expected from reusable, domain specific architectures and components that are of sufficient size to be worth reusing. Thus, we are currently focusing on the reuse allowed by object-oriented frameworks for this purpose [Lewis95].

Since a framework provides a pre-defined class hierarchy, object interaction, and thread of control, developers must fit their applications into the framework. This means that in framework-based development, the static structure and dynamic behavior of the framework must first be understood and then adapted to the specific requirements of the application. It is assumed that the effort to learn the framework and develop code within the system is less than the effort required to develop a similar system from scratch. Although it is recognized that the effort required to learn enough about the framework to begin coding is high [Booch94], [Pree95], [Taligent95], little work has been done in the way of minimizing this learning curve.

3.1 White-Box Frameworks

We are studying the process of learning such a framework (or more generally, any unfamiliar system) and developing constructive reading techniques that may minimize the effort expended on program understanding in particular situations. This experiment involves the study of a *white-box framework*, which defines a set of interacting classes, usually abstract classes, that capture the invariants in the problem domain. Since the source code of the classes is accessible to the programmer, a white-box framework can be specialized by deriving application-specific classes from the base classes through inheritance and by completing or overriding their methods [Johnson,Foote88], [Schmid96]. Learning to use a white-box framework is the same as learning how it is constructed because the user must have detailed framework code knowledge.

We have defined two reading techniques for using a white-box framework to build new applications: a system-wide reading technique and a task-oriented reading technique. Both techniques look at the static structure and the run-time behavior of the framework, and both have access to the same sources of information. The main difference is the focus of the learning process: the system-wide technique focuses more on the big picture than on the detailed task to be accomplished (which is the focus of the task-oriented technique).

With the system-wide reading technique, programmers attempt to gain a broad knowledge of the framework design. As a consequence, they deliver the functionality required by the new application mainly by specializing the abstract classes of the framework. With the task-oriented reading technique, programmers use existing framework-based applications as examples and attempt to gain a specialized knowledge of the parts which are directly relevant for the required system. As a consequence, they deliver the functionality required by the new application mainly by changing the concrete classes of the examples.

To compare these two techniques we have conducted a repeated-project experiment, in which we present graduate students and upper-level undergraduates with an application task to be developed using the white-box framework ET++ [Lewis95]. ET++ is a sophisticated framework that poses learning problems which can be major inhibitors against its use. The overall goal of the experiment is to compare the reading techniques for framework understanding (system-wide and task-oriented) with respect to their effect on ease of framework learning and usage, i.e., the ease with which the framework is understood and functionality is added. Students receive separate lectures on the reading techniques and work in teams of three people. One half of the class has been taught the system-wide reading technique and the other half the task-oriented reading technique. Preliminary results show that

- Even a relatively well-designed although poorly documented framework presents many difficulties in learning how to derive framework-based applications
- Students demonstrated an overhead in learning the framework with high levels of frustration in the early weeks because of the investment in time without an immediate payoff in programming
- Students found it easier to learn in the beginning by reading and reusing example applications than by trying to first gain a comprehensive knowledge of the framework
- Difficulties were encountered with the system-wide technique because the documentation provided was at an insufficient level of detail to be useful, and because the technique gave little guidance as to which area of the framework to concentrate on first.
- Difficulties were encountered with the task-oriented technique because it was hard to find suitable examples for all required functionality and because example applications were sometime inconsistent in terms of structure and organization.

3.2 Black-Box Frameworks

Black-box frameworks allow an application to be created by composing objects rather than by programming [Johnson, Foote88], [Schmid96]. They provide alternative concrete classes which have to be selected when creating an application, allowing some variability in the applications created. Thus, a black-box framework is customized by selecting, parameterizing, and configuring a set of components that provide the application specific behavior.

The interface between components can be defined by protocol, so the user needs to understand only the external interface of the components. Since this does not require knowledge of the framework code, black-box frameworks could be considered easier to use than white-box frameworks. However, better documentation and training are required because developers cannot look at the source code to determine what is going on.

We intend to investigate reading techniques for black-box frameworks in a real development context, focusing on the Generalized Support Software (GSS), a black-box framework developed and used to enable much more rapid deployment of flight dynamics applications at NASA/GSFC. The process for configuring a new mission-support application with GSS consists of selecting GSS classes to compile and link together, and setting values for a large number of control and operational parameters. The size and sophistication of the reuse asset library poses learning problems which can be major inhibitors against its use. Here, the goal is to improve the existing reading techniques which are used to understand which generalized components must be configured in order to develop new applications.

Variations of the two reading techniques that were compared in the ET++ experiment (system-wide and task-oriented) will have to be designed for use with a black box framework. The idea behind each reading technique will be the same, however. One will require the framework user to learn the overall structure of the framework, while the other will help the student learn with specific examples.

4. Conclusions

Much of our work in reading has so far focused on three families of reading techniques:

1. the defect-based reading family for analyzing requirements specification written in SCR notation, with the purpose of defect detection;
2. the perspective-based reading family for analyzing requirements specification written in English language, with the purpose of defect detection;
3. the scope-based reading family for constructing applications through reuse of white-box frameworks.

We will continue to conduct empirical studies which will allow us to closely monitor the use of different reading techniques in laboratory and real projects, both quantitatively and qualitatively. We believe it is necessary to integrate results from both types of studies in order to gain a deeper understanding of the research questions.

As our ability to understand software reading as a technique evolves, we plan to develop other families of reading techniques parameterized for use in different contexts and empirically evaluated for those contexts..

References

V. Basili, S. Green, “Software process evolution at the SEL”, *IEEE Software*, pp.58-66, July 1994.

V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Soerumgaard, M. Zelkowitz, “The empirical investigation of perspective-based reading”; *Empirical Software Engineering - An International Journal*, vol. 1, no. 2, 1996.

V. Basili, R. Selby, “Comparing the effectiveness of software testing strategies”, *IEEE Transactions on Software Engineering*, vol. SE-13, no. 12, pp.1278-1296, December 1987.

G. Booch, , “Designing an application framework”, *Dr. Dobb's Journal*, vol. 19, no. 2, February 1994.

R. Johnson, B. Foote, “Designing Reusable Classes”, *Journal of Object-Oriented Programming*, June 1988

T. Lewis et al., *Object-Oriented Application Frameworks*, Manning Publications Co., 1995.

A. Porter, L. Votta, V. Basili, “Comparing detection methods for software requirements inspections: a replicated experiment”, *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp.563-575, 1995.

W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley Publishing Co., 1995.

H. Schmid, “Creating Applications from Components: A Manufacturing Framework Design”, *IEEE Software*, vol. 13, no. 6, pp. 67-75, November 1996.

R. Selby, V. Basili, Baker, “Cleanroom software development: an empirical evaluation”, *IEEE Transactions on Software Engineering*, pp.1027-1037, September 1987.

Taligent Inc., *The Power of Frameworks*, Addison-Wesley, 1995.