Read Before You Write

Thomas B. Hilburn, Massood Towhidnejad, Salamah Salamah
Embry-Riddle Aeronautical University
hilburn@erau.edu, towhid@erau.edu, salamahs@erau.edu

Abstract

This paper describes and advocates a focused approach to using inspections of software artifacts as an active learning technique in software engineering education. A central thesis is that one must "learn to read before they write"; that is, you should read and study an existing software artifact, before you develop one. There is discussion of how software artifacts and supporting instructional materials from a Digital Home case study project can be used to support and guide software inspection exercises. These inspection exercises are designed to introduce students to realistic software engineering artifacts and involve them in rigorous examination of their contents. Instances of the use of software inspections to teach software engineering are described and analyzed: the experiences of students and instructors, what worked and what did not, and how this influenced the cases study project. The authors also outline a set of topics and courses in which software inspections might be used as a teaching tool throughout a computing curriculum.

1. Introduction

In the last twenty years there has been considerable interest and research in applying "active learning" techniques and activities to improve student learning. Students work in teams; they study and analyze materials, sometimes role-playing; they use case studies to evaluate and debate key concepts and techniques; and they engage in cooperative learning [1]. The teacher becomes a facilitator, guiding and coaching, rather than directing and lecturing. Students learn by doing rather than being given detailed structured techniques for solving problems.

Active learning can involve reviewing existing engineering artifacts and answering questions or solving problems associated with them. Often the review itself (rather than answers or solutions) has a greater positive effect on learning. For example, an active learning exercise in a computer programming class might involve reviewing program code and assessing its quality: Is the code correct?, Is it readable?, Is it maintainable?, Does it adhere to a coding standard? Deimel and Naveda [2] argue that the ability to read and understand a computer program is a critical skill for the software developer. They argue that teachers should undertake activities designed to teach and improve the program reading skills of their students. Weinberg, Kernighan and Plauger, and Knuth [3, 4, 5] have also promoted program reading - the central tenet is "read and understand programs before you write them". McMeekin et al [6] conducted research which showed the value of code reviews in building a student's critical software reading and analysis skills.

Reading is an essential and fundamental concept in teaching students how to write in a natural language. In the early grades we concentrate on learning to read – we build a reading vocabulary, we learn about language syntax and structure, and we begin to appreciate idiom and style, and how words and phrases are assembled to communicate information and meaning. As we proceed through school and the rest of life, we iterate between reading and writing, improving our capability in both areas. In this paper we advocate a similar active learning approach in software engineering education: read and review software artifacts

978-1-4577-0348-5/11/$26.00 © 2011 IEEE CSEE&T 2011, Waikiki, Honolulu, HI, USA

(requirements and design specifications, code, test plans, etc.) before you develop them; that is – *Learn to Read Before You Write*.

1.1. Software Inspections

It is now well understood and accepted that software quality is a major influence on software cost and schedule, and on the ultimate success of a software product. Experience through the years has shown that quality cannot be an afterthought, but must be "engineered" into the product throughout the software lifecycle. Although the use of proven techniques and tools for project planning, requirements engineering, design, and construction can significantly reduce the injection of defects, human error and lack of capability play a significant role in the defect density of development artifacts. The most influential factor in addressing quality, cost, and schedule issues is the early removal of defects; removing defects in the early phases of development can reduce the time and cost of defect removal by 10 to 100 fold, as compared to removal in test or in the field [7, 8].

One of the most effective, proven techniques for defect removal is software inspection. In [7], Fagan states that software inspections at IBM have "enabled higher predictability than other means, and the use of inspections has improved productivity and product quality". Parnas and Lawford [9] assert that "In addition to finding errors in code and related software documents, inspection can also help to determine if coding style guidelines are followed, comments in the code are relevant and of appropriate length, naming conventions are clear and consistent, the code can be easily maintained, etc.". Although Fagan inspection techniques are used widely in industry, there are still many in the commercial and academic computing communities who do not understand their value, misuse the technique, or are unaware of the technique.

By a software inspection we mean a structured formal review of a software artifact by a team of software developers (or students). Fagan described an inspection process that included defined roles (author, moderator, recorder, inspector), advanced preparation, inspection, rework, follow up, and collection and analysis of inspection data. We describe such a process in section 3 of this paper.

We advocate the use of Fagan inspections as an excellent way to teach software engineering: all engineering fields use some form of inspection (typically a formal design review); it is a team activity; it involves emersion in engineering artifacts; it requires technical knowledge to participate; and it involves measurement and use of data to analyze and draw conclusions. We also show that such inspections can be structured as active learning exercises.

1.2. Case Study Teaching

Many computing programs have a software engineering course that involves a software development project in which students are grouped into teams to work on a semester or year-long software development project. Unfortunately, this is too often isolated from the rest of the curriculum and does not form a real-world basis for the entire curriculum [10]. As a result, these programs produce graduates who are familiar with the basic theoretical concepts in software development, but lack the skills to apply these concepts in real-world development environments. Therefore, it is imperative that computing curricula introduce professional and real-world education into the academic programs. The use of case studies is one widely-used method for introducing real-world professional practices into the classroom.

Case studies were first used in the Harvard Law School in 1871 [11]. Since then, case studies have been a subject of much study and research about their effectiveness in teaching

and learning [11, 12, 13, 14]. They have become a proven and pervasive method of teaching about professional practice in such fields as business, law, and medicine. The term "case study" is used in a variety of ways. In its most naive form, it simply refers to a realistic example used to illustrate a concept or technique. More formally, a case study involves the application of knowledge and skills, by an individual or group, to the identification and solution of a problem associated with a real-life situation. Such a case study would contain an account (often in a scenario format) of a real-world activity, event, or situation. It might be supported with background material (setting, personalities, sequence of events, and problems and conflicts), artifacts, and data that are relevant to the situation depicted. The authors have used case studies in a variety of software engineering education activities: undergraduate and graduate software engineering courses, faculty workshops, and industrial short courses.

Case studies are active learning tools; they are meant to encourage participation, debate and understanding. Although they can be used in a didactic, teacher-centered pedagogy, they are most often used in an active learning, student-centered system where the teacher acts as a facilitator or "Socratic" coach. There are many excellent sources for developing and using case studies in an active learning manner – for example:

- The National Center for Case Study Teaching in Science at the State University of New York at Buffalo (http://ublib.buffalo.edu/libraries/projects/cases/case.html), and
- The Center for Teaching Excellence at the University School of Dentistry and Medicine of New Jersey (http://cte.umdnj.edu/active_learning/active_case.cfm).

2. The Digital Home Case Study

For several years the authors have been involved in a case study project that focused on the development of a Digital Home system [15, 16]. The Digital Home Project, when completed, will cover the complete life-cycle development of a software product (project management, requirements analysis and specification, design, implementation, testing and maintenance). The initial phase of the case study project concentrated on building a foundation for full development: research into case study teaching; identifying a case study problem; creating a scenario framework; describing the launch of the software development team; fashioning a software development plan to guide development of the Digital Home System; establishing a development process; creation of a Digital Home need statement; analysis, modeling and specification of the Digital Home requirements; development of a system test plan; development of a software architecture; and specification of the system components.

We have also developed a set of case study exercises (we call "case modules") which are designed to engage students in active learning software engineering activities related to development of the Digital Home System. Existing Digital Home scenarios, artifacts and case modules can be viewed at http://www.softwarecasestudy.org/.

As described in a case study introductory scenario, the DigitalHome project is part of the vision of the future for a national company HomeOwner, which is the largest national retail chain serving the needs of home owners in building, furnishing, repairing, and improving their homes. The HomeOwner management has decided to develop a prototype DigitalHome (DH) system that has the following features:

- The DH system will allow any web-ready computer, cell phone or PDA to control a home's temperature, humidity, security devices, and the state of small appliances.
- The communication center of the DH system will be a personal home owner web page, through which a user can monitor and control home devices and systems.
- The Digital Home will contain a master control device that connects to the home's broadband Internet connection, and uses wireless communication to send and receive communication between the DH system and the home devices and systems.

- The Digital Home will be equipped with various environment sensors (temperature sensor, humidity sensor, power sensor, contact sensor, etc.). Using wireless communication, sensor values can read and saved in the home database.
- The DH system includes programmable devices (thermostats, humidstats, security devices, and small appliance power switches), which allows a user to easily monitor and control a home's environmental characteristics from any location, using a web ready device.
- The DH system includes a DH Planner, which provides a user with the capability to direct the system to set various home parameters (temperature, humidity, lighting, and on/off appliance status) for certain time periods.

The DH scenarios describe a set of characters that are involved, directly and indirectly, in the development of the DH system. These include the following:

- HomeOwner management personnel – Robert "Red" Sharpson (founder and CEO), Dick Punch (VP of Marketing) and Judy Fielder (Chief Information Officer).
- A family used in the DH needs assessment – the Wrights (Steve, Mini, Stanley, Vinni, Michelle and Robert)
- The DH development team – a diverse team of professionals with a variety of experience and background: Jose Ortiz (Director, DigitalHomeOwner Division), Disha Chandra (Team Leader), Michel Jackson (System Analyst), Yao Wang (System Architect), Georgia Magee (Software Engineer), and Massood Zewail (Software Engineer). The team represents a diverse group of "real people", who bring different talents and experiences to the team. In the case study, this variety is explored and exploited in bringing realistic learning situations to students. (Team bios can be viewed at http://www.softwarecasestudy.org/.)

3. A Software Requirements Inspection Case Module

An early Digital Home case study effort was the analysis and specification of the software requirements for the system, which resulted in a Digital Home Software Requirements Specification (SRS). In addition to the specification of the functional requirements, the SRS includes a description of user characteristics, development constraints, the performance environment, and nonfunctional requirements specifying performance, reliability, and safety and security requirements. The SRS also includes a use case model. A copy of SRS version 1.3 can be downloaded at http://www.softwarecasestudy.org/.

3.1. Case Module Description

As part of the development of the requirements specification, the team developed a case module for inspection of the SRS. Tables 1, 2 and 3 provide a description of various case module elements. The details of some elements are not included for brevity's sake.

Table 1 provides an overview of the contents of a module. The learning objectives for this case module go beyond assessing the quality of the SRS, but are intended to address critical software engineering education goals: appreciating and understanding the problems in specifying requirements; learning to work as part of a team; and using and following an inspection process. Notice the list of DH artifacts and the contents of the inspection package. Although the SRS artifact is the chief focus of the inspection, the others (such as the background scenario and need statement) provide the setting and context to create a realistic environment for a professional and effective software

inspection exercise. The inspection package provides the sort of forms and tools used in a mature inspection process and help guide the students not only in performing an effective inspection, but in understanding how a best practice works.

Table 1: SRS Inspection Module Outline

Case Module: SRS Inspection
Prerequisite Knowledge: Understanding of basic elements of a Fagan Software Inspection process.
Learning Objectives: Upon completion of this module students will have increased ability to: 1. Work as a member of an Inspection Team 2. Assess the quality of a Software Requirements Specification(SRS) 3. Describe problems in specifying the requirements for a software product. 4. Work more effectively as part of a team. 5. Explain the inspection process. 6. Describe the value of the Fagan inspection process.
Keywords: Customer Needs, Software Requirements Specification, Fagan Software Inspection
Case Study Artifacts: 1. DH Customer Need Statement 2. DH High Level Requirements Definition (HLRD) 3. DH Background Scenario 4. DH Team Biographical Sketches 5. DH SRS, Version 1.2 **Inspection Package:** • Inspection Process Description • SRS Inspection Checklist • Defect Log • Inspection Summary Report Form
Case Study Participants: …
Scenario: …
Exercise: …
Appendices: Exercise Booklet
Resource Information: …
Teaching Notes: …

Table 2 includes a scenario that is part of the SRS Inspection Case Module. Scenarios are used throughout the Digital Home life-cycle to provide development activities to provide context and some realism to the learning activities associated with Digital Home.

Table 3 provides a set of notes intended to support use of case modules in undergraduate and graduate software development courses, providing guidance and suggestions to a teacher using the case module.

3.2. Class Use of the Inspection Module

The SRS Inspection Case Module (and variations of it) has been used in a variety of courses and workshops: a sophomore-level introductory course in software engineering; an undergraduate course in software quality assurance; an undergraduate course in software

requirements; a graduate course in software architecture; a faculty workshop in software process; and a short course in software reliability.

Table 2: SRS Inspection Module Scenario

Case Module: SRS Inspection
Case Study Participants: • The DH Team & Jose Ortiz, Director, DigitalHomeOwner Division of HomeOwner, Inc.
Scenario: In early September of 2010, HomeOwner Inc. (the largest national retail chain serving the needs of home owners) established a new DigitalHomeOwner division that was set up to explore the opportunities for equipping and serving "smart houses" (dwellings that integrate smart technology into every aspect of home living). … The Marketing Division produced two documents: the *DH Customer Need Statement* and the *DH High Level Requirements Definition* (HLRD). In September 2010, a five person team was assembled for the project and in early October 2010 carried out a "project launch". After project planning was completed the team began work on requirements analysis and specification. The first version, 1.0, was completed in early October and versions 1.1 and 1.2 were completed by mid October. In consultation with Jose Ortiz, the team has decided to carry out a formal Fagan inspection of the SRS, version 1.2. Jose has agreed to act as a customer on the inspection team, Michel Jackson is the author, Disha Chandra will be the moderator and other roles will be assigned in the overview meeting.

Table 3: SRS Inspection Module Teaching Notes

Case Module: SRS Inspection
Teaching Notes: • This case module could be used in different level courses (from an introductory level course in software engineering to an upper level or graduate course in requirements engineering or quality assurance.). • Assuming an adequate student preparation for the exercise, allowing students about three hours each for the exercise should be sufficient: assuming two hours for preparation and inspection, and one hour for the inspection meeting. • Preparation time can be done as an outside class activity to be reported as part of a deliverable before inspection meeting • It would be beneficial to follow the exercise with a twenty to thirty minute discussion concerning the student team results. Some key points to include in the discussion are the following: ➢ Discuss how closely the inspection process was followed: ➢ How well did the team conduct each phase? ➢ How well did students carry out their assigned inspection role (i.e., moderator, author, inspector)? ➢ … ➢ … ➢ Address any personal conflict or egoistic attitude displayed. • Student team members should be cautioned about a few things: ➢ Leave their ego outside of the meeting room ➢ Their job is to identify defects, not fix them. ➢ …

In fall 2009 and spring 2010 the SRS Inspection Case Module was used in three software engineering classes (two sophomore level classes and one senior level class), in which nine inspection teams were formed to carry out the module exercise. The inspections concentrated on the functional requirement statements and used the Inspection Package described in Table 1: Inspection Process Description, SRS Checklist, Defect Log, and Inspection Summary Report Form. The Report Form includes summary data about the inspection. The inspection process and materials are based on the work of Fagan [6] and Humphrey [17].

Although all teams followed the inspection process, and collected and recorded inspection data, some elements of the process were performed weakly and not all data was accurate or complete. Table 4 shows the data for one team, which could be considered typical. In the comments section of Table 4 we compare the team's results with benchmark goals for inspection data (based on work by Humphrey [15]). Although the defect removal rate and the overall inspection rate seem reasonable, the team identified only 10 major defects, while the DH project team had previously reviewed the SRS and identified over 20 major defects (many purposely seeded in the SRS). Two types of defects commonly missed were compound statements containing multiple requirements, and the testability problems with some requirements statements. One conclusion might be the inspection team was ineffective; however, we viewed this more as an education exercise, not strictly a quality assurance activity. By engaging students in a "close" reading of the SRS, we helped them to understand its meaning, to determine the degree to which it addressed the customer need statement, to evaluate the correctness, clarity and precision of the requirements statements, and to identify missing features. We should also note that the collection and analysis of team inspection data provides the teacher with excellent information on how the inspection was conducted and the degree to which the case module objectives were reached.

Table 4: SRS Inspection Team Data

Inspection Features	Team Data	Comment
Requirements Size	7 pages	The team only reviewed a portion of the SRS.
Major Defects Identified	10	Correction of a major defect either changes the program source code or would ultimately cause change in the program source code.
Major Defects Missed	2	Total defects of 12 were estimated using the "capture-recapture" method [17]
Total Inspection Time	14.2 hrs	9.2 hrs for preparation time and inspections time; one hr inspection meeting (times 5 people).
Defect Removal Rate	0.7 def/hr	A benchmark goal is 0.5 def/hr [17]
Inspection Rate	0.49 pg/hr	A benchmark goal is < 2 pg/hr [17]

We believe this sort of "reading" is a critical, first step in helping students (or professionals) prepare for requirements analysis and specification tasks. Anyone that has been an observer in a student inspection meeting has seen the degree to which a structured inspection activity engages students in disciplined reading and understanding of the inspection artifact: students discuss and debate the correctness, understandability and completeness of artifact elements; they reach an understanding of what the artifact should and should not contain; and they get a real sense of the meaning of "quality". In [6], research was conducted on the utterances of students while engaged in code inspection teams: the research demonstrated various levels of learning beyond simple knowledge of the code content. This is

a result similar to what we observed in the SRS inspections and that was illustrated in the defect logs and the inspection summary reports.

Of course, inspecting an SRS does not make one a requirements analyst, but it is an important first step; and more importantly it promotes the sort of understanding of what requirements are, how difficult is to get them right, and why they are important. This sort of learning is essential to any student who aspires to be a professional software developer.

The experiences of inspection teams also provided valuable input to the DH project team of how well the SRS Inspection Case Module worked. In analysis of the overall inspection experience we did see ways in which the case module could be improved. We reviewed all inspection forms to insure they were understandable and convenient to use and we verified that we had included checks for compound statements and requirement testability in the inspection checklist. This motivated several changes to the module: revision of the inspection process and the SRS Checklist to improve clarity and completeness, and addition of a comment in the Teaching Notes and Exercise Booklet about the need to emphasize the importance of properly collecting and recording inspection data. We also think a requirements tutorial offered before the use of the case module would be helpful, especially for beginning students. The tutorial could include the following: the purpose and importance of an SRS; the typical structure and content of an SRS; a description of the inspection process, data, and forms; and examples of well-written and poorly written requirements.

Table 5: A Curriculum Inspection Framework

Inspection Artifact	Course(s)*
Project Plans	SE323 Software Project Management
Risk Management Plan	SE323 Software Project Management
Software Process Description	SE324 Software Process and Management
Requirements Specification	SE322 Software Requirements Analysis
User Interface Specification	SE212 Software Engineering Approach to HCI
Traceability Matrix	SE322 Software Requirements Analysis
Architectural Specification	SE311 Software Design and Architecture
Module Specification	SE311 Software Design and Architecture
Algorithm Design Specification	SE211 Software Construction
Code	SE211 Software Construction
Unit Test Plan	SE211 Software Construction
Integration Plan	SE311 Software Design and Architecture
System Test Plan	SE321 Software Quality Assurance and Testing
All of the Above	SE101 Introduction to Software Engineering SE400 Software Engineering Capstone Project
* Course number and names are from [18]	

4. Comprehensive Use of Inspections

We believe that the inspection process described in Section 3 can be applied to almost any artifact that is associated with the software life-cycle. As was mentioned in Section 2 the DH Case Study project is developing a set of artifacts that cover all phases of the life-cycle. Using software inspection as a curriculum-wide teaching technique, inspection of DH artifacts (or artifacts from other case studies) could be used in a variety of software engineering and computing courses. Table 5 lists example artifacts and courses (selected from [18]) for which a comprehensive inspection framework could be implemented. Of course, there are other courses and artifacts that could be used, and the row labeled “All of the Above” lists courses

which cover the entire life cycle and for which a variety of artifacts might be inspected. It should be noted that because of the differences in student maturity and background, the level and quality of the results would differ in SE101and SE400.

An important side effect of this approach of using inspection as a curriculum-wide pedagogy is the emphasis on the importance of a quality focus (and its corollaries of cost and schedule) throughout the software lifecycle. We believe that the use of inspections early in curriculum (in the very first programming course) can provide the sort of professional experience that does not have the same positive influence as if introduced late in the curriculum – much like early defect removal increase quality and productivity. Also, the use of numerous inspections which involve measurement, analysis of measured data, and the use of derived metrics to assess both product and process quality help students develop an "engineering" mindset.

5. Final Thoughts

Software inspection is not only a proven software engineering best practice, its use in the classroom is an effective active learning technique. If carried out properly, an inspection requires close reading and study of a software artifact; it requires the inspection team to not only understand what the artifact contains (in a technical sense), but it requires the team to make a judgment about whether the artifact carries out its purpose and how well it does this. This marriage of the two uses of inspection (as a professional practice students learn to use and as a tool to teach about software development) reinforces the value of both.

6. Acknowledgements

Initial work on the DigitalHome case study was funded as part of the NSF project: "The Network Community for Software Engineering Education" (SWENET) (NSF 0080502. The current work on the case study is funded through NSF's (DUE- 0941768) "Curriculum-wide Software Development Case Study".

7. References

[1] C. Bonwell, and J. Eison, Active Learning: Creating Excitement in the Classroom AEHE-ERIC Higher Education Report No. 1, Washington, D.C,: Jossey-Bass, ISBN 1-87838-00-87, 1991.

[2] L. E. Deimel and J.F. Naveda, Reading Computer Programs: Instructor's Guide and Exercises, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-90-EM-3, August 1990.

[3] Gerald M. Weinberg, The Psychology of Computer Programming. New York: Van Nostrand Reinhold, 1971.

[4] Brian W. Kernighan and P. J. Plauger, The Elements of Programming Style, 2nd Ed., New York: McGraw-Hill, 1978.

[5] Donald E. Knuth, "Literate Programming." Computer, Vol 27, No 2 , May 1984, pp. 97-111.

[6] D. A. McMeekin, B.R. von Konsky, E. Chang, and D.J.A. Cooper, "Evaluating Software Inspection Cognition Levels Using Bloom's Taxonomy", 22nd Conference on Software Engineering Education and Training, Hyderabad, India, February 17 - 19, 2009, pp. 232-239.

[7] Michael E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Vol 15, No 3, 1976, pp. 258-287, pp. 744-751.

[8] Michael E. Fagan, "Advances in Software Inspections", IEEE Transactions in Software Engineering, Vol 12, No 7, July 1986.

[9] D.L. Parnas, and Mark Lawford, "The Role of Inspection in Software Quality Assurance", IEEE Transactions on Software Engineering, Vol. 29, No. 8, August 2003, pp. 674-676.

[10] S. Ludi, S. and J. Collofello, "An Analysis of the Gap between Knowledge and Skills Learned in Academic Software Engineering Course Projects and Those Required in Real Projects", Thirty-First ASEE/IEEE Frontiers in Education Conference, October 2001, pp. TD8-TD11.

[11] Ann M. Tomey, "Learning with Cases", Journal Of Continuing Education In Nursing, Vol 34, No 1, January/February 2003.

[12] Davis, C. and Wilcock, E., "Teaching Materials Using Case Studies", UK Centre for Materials Education, http://www.materials.ac.uk/guides/casestudies.asp, accessed November 2010.

[13] Fritz H. Grupe and Joelle K. Jay, "Incremental Cases", College Teaching, September 22, 2000, pp. 123-128.

[14] C. F. Herreid, "Case Studies in Science: A Novel Method of Science Education", Journal of College Science Teaching, February 1994, pp. 221-229. (http://ublib.buffalo.edu/libraries/projects/cases/teaching/novel.html, accessed November 2010)
[15] M. Lutz, G. Hislop, T. Hilburn., W. M. McCracken, and M. Sebern, "The SWENET Project: Bridging The Gap From Bodies Of Knowledge To Curriculum Development", Proceedings of 2003 Frontiers in Education Conference, Bolder, CO, November 2003.
[16] T. Hilburn, M. Towhidnejad, S. Nangia, and L. Shen "A Case Study Project for Software Engineering Education", Proceedings of 2006 Frontiers in Education Conference, San Diego, CA, October 2006, pp. TIA1-TD1A5
[17] Watts S. Humphrey, Introduction to the Team Software Process, Addison-Wesley, 2000.
[18] IEEE Computer Society (IEEE-CS) & the Association for Computing Machinery (ACM). ―Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.‖ Computing Curriculum Series, 2004. (http://sites.computer.org/ccse/SE2004Volume.pdf, accessed November 2010)