# Empirical Software Engineering in Industry Short Courses

David S. Janzen, Clark S. Turner
*Computer Science Department*
*California Polytechnic State University*
*San Luis Obispo, CA USA*
{*djanzen,csturner*}*@csc.calpoly.edu*

Hossein Saiedian
*Electrical Engineering and Computer Science Department*
*University of Kansas*
*Lawrence, KS USA*
*saiedian@eecs.ku.edu*

**Abstract**

*This paper reports on a pilot project that incorporated small empirical studies in three industry short courses. These laboratory experiments were one component of a larger leveled study on the effects of test-driven development (TDD) on internal software quality.*

*The approach is proposed to have pedagogical value to student-developers by improving their understanding and appreciation for empirical evidence, to instructors by providing feedback through surveys and exercises, and to the community at large by reporting results of the studies.*

*Pre-experiment surveys in the three pilot experiments revealed large differences in programmer opinions of TDD. Possible correlations to development environment and programmer experience will be proposed. Post-experiment surveys revealed improvements in programmer opinions of TDD following the experiment exercises.*

*Crafting sufficiently small but interesting assignments proved to be challenging. Few complete solutions were submitted and some developers were unwilling to submit their partial solutions.*

*Positive observations will be made regarding the use of experiments in short courses. For instance, participating in the study encourages analytical thinking, prompts developers to evaluate alternative approaches, and instills the value of empirical evidence. Ethical concerns regarding threats to validity are raised and addressed. The authors find that ethical considerations not only support performing such studies, but encourage it as the duty of software professionals.*

## 1. Introduction

Evidence-based software engineering (EBSE) endeavors to produce a body of documented experiences that might inform software practice adoption decisions. Evaluative research methods such as case studies and controlled experiments are commonly employed to demonstrate the efficacy of software practices, tools, and methods. Evidence-based paradigms have proven successful in disciplines such as medicine. Many anticipate similar benefits from widespread adoption and appreciation of an evidence-based approach in software engineering [7].

Despite increased interest in evidence-based software engineering [2], the growth of EBSE research is somewhat slow [5] and difficult. Many factors contribute to the challenges of EBSE, not least among them are access to conduct field experiments. While laboratory experiments are often conducted in academic environments, there are many inherent threats to validity. Students are rarely as mature as professional software developers. Application domains are often contrived. Software projects are rarely as large and complex as "real-world" projects.

Unfortunately companies and organizations are often reluctant to participate in field experiments. Many may be unwilling to try new, perhaps unproven approaches. Others may be concerned that they might reveal poor metrics or performance. Or they may be unwilling to allow researchers in for fear of losing proprietary information or simply that they may slow down the team. Regardless the reasons, barriers must be overcome in order for EBSE to advance.

Education on EBSE is proposed as a possible strategy to reduce entry barriers for conducting field experiments. Professional developers often acquire new skills through professional training. We propose that small laboratory controlled experiments can be integrated into many training courses. By obtaining first-hand experience participating in a short controlled experiment, it is believed that student-programmers will gain an appreciation for EBSE, they will gain analytical skills for comparing approaches, and they will be more open to allowing larger field experiments within their organization. The goal is not to train the industry student-programmers to conduct their own experiments, simply to raise awareness of EBSE among industry practitioners. This approach may help satisfy the need for more new and replicated studies [1], as well as satisfy an ethical duty of software professionals to assist colleagues and develop the field.

This paper describes the authors' experience with three laboratory experiments in professional training courses. The experiments were designed as part of a larger set of leveled experiments considering the internal software quality effects of test-driven development. Twelve leveled experiments were conducted in academic and professional settings from introductory programming through graduate software engineering courses, and in field and laboratory experiments with professional developers. The experiments compared an iterative test-first approach with an iterative test-last approach by analyzing numerous software metrics in the general categories of software size, complexity, coupling, cohesion, and testing. The results of the experiment will be reported here primarily for example purposes. The primary goal of this work is to expose industry practitioners to EBSE techniques so they can better understand EBSE results when making adoption decisions, and to encourage industry willingness to participate in larger EBSE studies.

Pedagogical value will be discussed, along with the threats to validity and ethical considerations of conducting and distributing results from such small laboratory experiments.

## 2. Course and Experiment Design

The lead author developed and presented three industry training courses to professional software developers in two Fortune 500 corporations. The first course introduced C++ to experienced C pro-

| Experiment | Timeframe | Students | Assignment | Pairs/ Solo | Submissions TF | Submissions TL |
|---|---|---|---|---|---|---|
| TDD in Java | Fall 2005 | 15 | Bowling | Both | 3 | 3 |
| C++ for C Programmers | Summer 2005 | 14 | Bowling | Solo | 2 | 4 |
| TDD in Java | Fall 2006 | 14 | Bowling | Pairs | 2 | 3 |
| TDD in Java | Fall 2006 | 14 | ToDo List | Pairs | 4 | 0 |

**Table 1. Experiment Profile**

grammers. This was a four-day course with a segment on test-driven development on the morning of the final day. The second and third courses introduced test-driven development to experienced Java programmers in two different companies. These were both two-day courses. All courses were delivered in full-day, on-site, lab-based environments. Course enrollments were 15, 14, and 14 respectively.

Course participants were provided basic instruction on automated unit testing and given short lab exercises to establish basic competency. The two Java courses utilized JUnit and the C++ course utilized simple assert statements due to the shorter time frame.

Participants were given a pre-experiment survey to measure developer experience and opinions, then instructed in both an iterative test-first and test-last development approach. Extensive discussion of test-driven development was delayed until after the experiment was completed and the post-experiment surveys were administered. Students were then divided into test-first and test-last groups and given a programming assignment.

The exercise was to build a bowling game scorer as described by Robert C. Martin [8]. The same exercise was used in an industry experiment by Laurie Williams [4] to examine the effects of TDD on external quality. The project involved reading bowling throws from a file, calculating scores, and presenting scores through a text-based user interface. Approximately two hours was given to complete the assignment. Some sample input/output code was provided to subjects to shorten the development effort.

In the third course, participants were also given a second programming assignment on the following day and asked to switch test-first/test-last approaches. The second programming assignment was a simple To-Do list planner. In this course, post-experiment surveys were administered after both programming exercises.

## 2.1. Experiment Results

Students were asked to submit code and tests from the four projects in the three experiments. Only between 42% and 73% of the students submitted their projects and a few of the submitted projects did not compile and/or were incomplete. Lack of time was the primary reason given for the low submission rate. Half of the Bowling projects received were completed with a test-first approach, but all of the To-Do list submissions were from test-first developers. Table 1 summarizes the submissions. Notice that the use of solo or pair programming is inconsistent. In the 2005 TDD course, a couple of individuals with limited Java experience requested to work in pairs with more experienced Java developers while others preferred to work solo. This was allowed and both test-first and test-last groups contained one or two such pairs of programmers.

| Experiment | Exercise | Approach | Coverage Line | Branch |
|---|---|---|---|---|
| TDD Summer 2006 | Bowling | TF | 81% | 79% |
| TDD Summer 2006 | Bowling | TF | 45% | 0% |
| TDD Summer 2006 | Bowling | TL | 91% | 87% |
| TDD Summer 2006 | Bowling | TL | 0% | 0% |
| TDD Summer 2006 | ToDo | TF | 100% | 100% |
| TDD Summer 2006 | ToDo | TF | 100% | 100% |
| TDD Summer 2006 | ToDo | TF | 88% | 75% |
| TDD Summer 2006 | ToDo | TF | 86% | 76% |
| TDD Fall 2005 | Bowling | TF | 50% | 19% |
| TDD Fall 2005 | Bowling | TF | 58% | 55% |
| TDD Fall 2005 | Bowling | TF | 49% | 30% |
| TDD Fall 2005 | Bowling | TL | 68% | 63% |
| TDD Fall 2005 | Bowling | TL | 73% | 80% |
| TDD Fall 2005 | Bowling | TL | 6% | 0% |
| | | Average TF | 73% | 59% |
| | | Average TL | 60% | 58% |

**Table 2. Test Metrics**

### 2.2. Software Metric Results

A suite of static metrics was calculated on the projects from the two Java experiments. Metrics were chosen to evaluate software size, complexity, coupling, and cohesion. Detailed discussion of the metric selection and results of the larger leveled study are available in [6]. Unlike the larger studies, in the training course studies no statistically significant differences existed between the software developed with a test-first and a test-last approach. Likely this is due to the small size of the projects completed.

Table 2 reports the test coverage metrics from the training experiment. Excluding the one project with no automated tests, the test-first projects had an average line and branch coverage of 73% and 59% respectively, compared with 60% and 58% for the test-last projects. The gap between test-first and test-last test coverage was even more significant in the leveled study with much larger projects.

### 2.3. Subjective and Evaluative Results

Surveys were conducted immediately before and immediately after the programming exercises in all three courses. No statistically significant differences existed between the test-first and test-last groups in terms of academic background, work experience, or specific programming experience.

Programmer responses on three questions were analyzed for changes from the pre to the post experiment survey. The questions rated programmer attitudes toward the following factors:

- importance of unit testing (Attitude)
- timing of writing unit tests (Timing)
- choice of test-first or test-last programming (Choice)

| Experiment | Direction | Attitude | Timing |
|---|---|---|---|
| TDD Fall 2005 | %Increasing | 62% | 50% |
| | %Decreasing | 0% | 0% |
| C++ Summer 2005 | %Increasing | 0% | 88% |
| | %Decreasing | 13% | 0% |
| TDD Summer 2006 | %Increasing | 9% | 100% |
| | %Decreasing | 9% | 0% |

**Table 3. Programmer Attitude Changes**

| Experiment | Choice | %Pre | %Post | %Difference |
|---|---|---|---|---|
| TDD Fall 2005 | Test-First | 67% | 83% | 17% |
| | Test-Last | 33% | 17% | -17% |
| C++ Summer 2005 | Test-First | 29% | 33% | 5% |
| | Test-Last | 71% | 67% | -5% |
| TDD Summer 2006 | Test-First | 60% | 82% | 22% |
| | Test-Last | 40% | 18% | -22% |

**Table 4. Programmer Choice Changes**

Table 3 presents the results of this analysis. %Increasing indicates that respondents increased their opinions of the importance of unit testing (Attitude) and the "earliness" of writing unit tests (Timing) between the pre and post experiments. Sixty-two percent of the programmers in the "TDD Fall 2005" experiment indicated that testing was more important after participating in the experiment, whereas programmers in the other two experiments actually thought testing was less important or had mixed opinion shifts. In all of the experiments, 50% or more of the programmers changed their opinions to favor earlier testing.

The final question asked programmers whether they would choose to use the test-first or the test-last approach. Table 4 reports the changes from the pre to post experiment survey in programmer choice. In all cases more programmers chose the test-first approach after completing the experiment. These results are consistent with those from the larger studies. It is interesting to note the significant difference in programmer willingness to adopt the test-first approach between the different courses. The C++ programmers were far less open to the test-first approach. One explanation might be the nature of the courses. The C++ course was primarily a language course with the experiment on the last day, whereas the Java courses were specifically focused on test-driven development with the experiment on the first day. It seems likely that students coming to a TDD course are more open to trying TDD than students in any other non-TDD specific course. Additional rationale could be the different development environment. The C++ programmers used primitive assert statements whereas the Java programmers used the more sophisticated JUnit framework.

## 3. Pedagogical Considerations

The inclusion of controlled experiments in industry training courses is proposed as a win-win situation. Coupling such experiments with course content can greatly enhance courses while introducing only minimal overhead, primarily that of conducting surveys. Professional student-programmers attending courses containing such experiments will gain valuable experiences beyond merely learning the course material. Most notably, student-programmers may benefit by:

- recognizing that alternative approaches exist
- learning to analyze and evaluate EBSE comparisons of alternative approaches
- gaining appreciation for and understanding of EBSE

Instructors and the software community at large may benefit by:

- collecting survey data
- obtaining experimental results for analysis and possible dissemination
- opening the door to conduct full-scale empirical studies in an industry domain

Depending on the nature of the study, data obtained may be of limited value due to common threats to validity. Training courses will typically have small sample sizes. Short time frames will generally limit the size of exercises so that they may not be representative of industrial projects. Further the very nature of training courses indicates that developers will likely be immature in their use of the particular tools, languages, or practices being examined. As a result, any publications resulting from such studies should clearly advertise their limitations.

Despite such validity threats, valuable information may still be obtained from such short experiments. Evaluations of pedagogical approaches or learning curves seem appropriate. Results from short course studies can be combined with replicated studies to gain confidence, or data may augment other results from diverse studies as seen in the TDD studies above.

## 4. Ethical Considerations

### 4.1. Human Subjects

Industry short courses such as this one clearly support the IEEE/ACM Software Engineering Code of Ethics ("Code") [9] section 7.02 in that we "[a]ssist colleagues in professional development" with the instruction in software testing. However, gathering data from human subjects may also involve other serious considerations. The Belmont Report [3] states that "[a]pplications of the general principles to the conduct of research leads to consideration of the following requirements: informed consent, risk/benefit assessment, and the selection of subjects of research."

Human subjects approval was obtained from the University of Kansas for the broad set of leveled experiments in this study. In the training courses, informed consent was obtained from the corporate manager who sponsored each course. Participants were verbally informed regarding the nature of the experiment and their right to not participate. All student-programmers chose to participate and they were assigned identification numbers so their surveys and software artifacts could be correlated while preserving privacy. In this way, the first requirement (informed consent) was satisfied.

The second requirement, risk analysis, is a minor consideration for studies like these. The only risks to participants could involve a waste of time or a breach of confidentiality leading to job consequences. The time taken was minimal (fifteen minutes) and was approved by management. The time could also be considered part of the instruction since relevant EBSE issues were addressed in detail. Confidentiality was built into the experiment design, no personally identifying information was attached to any information.

The third requirement is inapplicable in studies like this since we're not involved in any specific "benefit" to participation (such as a medical treatment) and the subjects are clearly directly related to the problem being studied.

### 4.2. Our Duty to Perform Such Studies

Not only have these pilot studies been performed in an acceptably ethical manner, the Code appears to make it the very duty of those involved in industry short courses to consider conducting pilot studies and publishing the results. The most pertinent provisions are 6.02 and 6.03 where software engineers (in the general sense) are to "[p]romote public knowledge of software engineering" and to "[e]xtend software engineering knowledge by appropriate ... publications."

An interesting and substantial side effect of performing such pilot studies can be the education of the student-participants in EBSE design, methods, tools, and an appreciation for gathering and interpreting data. The authors suggest EBSE instruction become an integral part of performing the study and sharing the results during short courses. Such information contributes to the strength of the all-important "informed consent" as the participants will gain the information needed to give reasoned consent. Note also that we've further met the duty under the Code to "assist colleagues" in their professional development. Not only do they learn about TDD in this case, but they gain knowledge and experience in EBSE.

## 5. Conclusions

Integrating controlled experiments into industry training courses is proposed to have pedagogical and intellectual merit while maintaining ethical integrity. Further, by raising awareness of evidence-based techniques, access to conduct experiments in the field is expected to increase, thereby broadening the body of evidence-based software engineering knowledge.

The approach was applied with a study on test-driven development in three training courses. While little was revealed regarding TDD's influence on internal software quality, results did support the propensity of test-first developers to achieve higher test coverage than their test-last counterparts. In addition, survey results indicated that programmers moved toward preferring earlier testing, and they were more open to a test-first approach after participating in the study. Discussion and critical analysis of such results in the training courses occurs naturally in such courses and confirms the value of the approach.

## References

[1] Victor R. Basili, Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. *IEEE Trans. Softw. Eng.*, 25(4):456–473, 1999.

[2] David Budgen, Stuart Charters, Mark Turner, Pearl Brereton, Barbara Kitchenham, and Stephen Linkman. Investigating the applicability of the evidence-based paradigm to software engineering. In *WISER '06: Proceedings of the 2006 international workshop on Workshop on interdisciplinary software engineering research*, pages 7–14, New York, NY, USA, 2006. ACM Press.

[3] The National Commission for the Protection of Human Subjects of Biomedical and Behavioral Research. Ethical principles and guidelines for the protection of human subjects of research, 1979.

[4] Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337–342, 2004.

[5] Robert L. Glass, V. Ramesh, and Iris Vessey. An analysis of research in computing disciplines. *Commun. ACM*, 47(6):89–94, 2004.

[6] David Janzen. *An Empirical Evaluation of the Impact of Test-Driven Development on Software Quality*. PhD thesis, The University of Kansas, August 2006.

[7] Barbara A. Kitchenham, Tore Dyba, and Magne Jorgensen. Evidence-based software engineering. In*ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 273–281, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education, Inc., 2003.

[9] ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices. Software Engineering Code of Ethics and Professional Practice, 1992.