On the Need for a Process for Making Reliable Quality Comparisons with Industrial Data

Laurie Williams
North Carolina State University, Department of Computer Science
williams@csc.ncsu.edu

Abstract

Many factors influence quality data obtained from industrial case studies making comparisons difficult. In this paper, two longitudinal industrial case study experiences are shared which illustrate the complications that can arise. The first is a case study of an IBM team that transitioned to the use of test-driven development. The primary quality measure was functional verification test defects normalized by lines of code. The second case study was performed with an Extreme Programming team at Sabre Airline Solutions. Both test defects and field defects were compared. In both case studies, differences existed which made the comparisons indicative but not absolute.

1. Introduction

Making cost-effectiveness comparisons of testing and other software development processes in industrial case studies present a number of methodological and technical challenges. The challenge that will be addressed in this paper is that of determining if the methodological and technical changes resulted in the production of a higher quality product. One means of making this comparison is through a measure of defect density (defects/lines of code) of the resulting products. The defect density metric can be analyzed (1) prior to releasing the product to the customer by measuring defects by the testing team(s) or (2) after a product has been in the field for a specified number of months. However, important questions must be answered before comparing defect density products, such as:

Is the internally-visible (found in test) defect density better because less testing took place?

Is the externally-visible (customer) defect density better because fewer customers are using the product?

Answering these questions requires that additional data is collected in the case study. The community could benefit from having a process for performing case study comparisons of software process techniques assessing the impact of the technique on product quality. The proposed process would outline the data that should be collected, the analysis that should be performed, and the details on how the results should be presented.

This paper describes two longitudinal industrial cases studies that were performed by the author and her research team. The first is a case study of an IBM team that transitioned to the use of test-driven development. The primary quality measure was functional verification test defects normalized by lines of code. The second is a longitudinal case study that was performed with an Extreme Programming (XP) [1] team at Sabre Airlines. Both test defects and field defects were compared. In both case studies, differences existed which made the comparisons indicative but not absolute. Both case studies would have benefited from having the proposed process.

Section 2 and 3 provide background on the IBM case and Sabre Airlines case studies, respectively. Section 4 suggests a possible approach that can be worked on as a research community. Section 5 presents a summary.

2. IBM Test-Driven Development

In this section, information is provided on the details of the IBM test-driven development case study and the complications that arose in making a quality comparison.

2.1 Case Study Overview

We conducted a year-long case study with an IBM software development group [7, 10] to examine the efficacy of the test-driven development (TDD) [2] practice as a means for reducing defects in a software-intensive system. With TDD, before implementing production code, the developer writes automated unit test cases for the new functionality they are about to implement. After writing test cases, the developers produce code to pass these test cases. The process is essentially "opportunistic" in nature [4]. A developer writes a few test cases, implements the code, writes a few test cases, implements the code, and so on. The work is kept within the developer's intellectual bounds because he or she is continuously making small design and implementation decisions and increasing the functionality at a manageable rate. New functionality is not considered properly implemented unless these new (unit) test cases, and every other unit test case written for the code base, successfully pass.

The IBM group develops mission-critical software for its customers in a domain that demands high availability, correctness, and reliability. "Essential" money [3] and customer relations are at risk for IBM's customers if the software is not available, correct, and reliable. "Discretionary" money [3] and convenience are at risk for the recipients of the computer-dependant service provided by the IBM product. In our case study, we quantitatively examined the efficacy of TDD as it relates to defect density reduction before a black-box, functional verification test (FVT) run by an external testing group after completion of production code.

This IBM group has been developing device drivers for over a decade. They have one legacy product which has undergone seven releases since late 1998. This legacy product was used as the baseline in our case study. In 2002, the group developed device drivers on a new platform. In our case study, we compare the seventh release on the legacy platform with the first release on the new platform. Because of its longevity, the legacy system handles more classes of devices on more platforms with more vendors than the new system. Hence, while not a true control group, the legacy software still can provide a valuable relative insight into the performance of the TDD methodology.

All participating IBM software engineers on both projects had a minimum of a bachelor's degree in computer science, electrical or computer engineering. A few had master's degrees. The seventh release legacy team consisted of five, co-located full-time employees with significant experience in the programming language of choice (Java and C++) and the domain. The new product team was made up of nine full-time engineers, five in a US location and four in Mexico. Additionally, some part-time resources for project management and for system performance analysis were allocated to the team. No one on the new team knew TDD beforehand, and three were somewhat unfamiliar with Java. All but two of the nine full-time developers were novices to the targeted devices. The domain knowledge of the developers had to be built during the design and development phases.

2.2 Quality Comparison

One of the most interesting findings of the case study was that the defect density of the code entering FVT/regression test appeared to be significantly better for the "new" system when compared with the legacy system. The new product appears to exhibit approximately a *40% lower defect density*. The severity distribution of the defects (faults) was essentially equal in the two cases.

However, some differences between the two projects necessitate a more in-depth comparison. To help understand the defect or fault density differences, we turn to some testing issues – specifically to the number of test-cases run. One thing to note is that if a device was supported by both the legacy and the "new product" code, the FVT test cases for that device were identical. A "substitutability" requirement for the new system was to "pass all the legacy system FVT tests." An identical FVT/regression exit criterion was used for both projects. This criterion identifies the percentage of FVT and regression test cases that must be attempted/passed and the percentage of defects that may remain unresolved based on the severity level.

However, it must be noted that in absolute terms, the legacy product was tested using about twice as many test-runs when compared with the "new product". The reason is device diversity. Specifically:

- The devices on the legacy product had to run on two platforms (Windows and Linux). The "new system" devices needed to work only on Linux. (`numberOfOS`)
- The legacy product worked on more hardware platforms than the "new product." Test cases needed to be re-run for each platform. (`numberOfSystemFamily`)
- For each class of device (e.g. the printer class of device), the legacy product supported more brands/models of devices. As a result the same set of tests was often run multiple times on various but perhaps similar devices. (`deviceClass, numberModels,TCforDevice`)

Also for each class of device there is a percentage of the test cases that are only run once because they were common for all devices. Hence, the number of test cases needed for a class of device could be reduced by this factor. (`commonTCFactor` is used to account for this effect)

The total number of test cases (TC) run on each product were approximated by the following formula:

$$\mathrm{TC} = \sum_{deviceClass} (numberModels * TCforDevice * commonTCFactor) * \mathrm{numberOfOS} * \mathrm{numberOfSystemFamily}$$

Table 1 illustrates the results. For the legacy system, both the factor `numberOfOS` and the factor `numberOfSystemFamily` were 2 or more. In the new product both of these factors were 1. Also more brands/models were supported by the legacy. This means that significantly more test cases needed to be run on the legacy code (requiring more FVT effort) than on the "new project" code to meet the same FVT/regression criteria. When test cases are repeated for multiple hardware/software platforms, these test cases often execute the same lines of code. This drives up the ratio of test cases per LOC for the legacy product. Similarly, these multiple executions drive down the ratio of defects to number of test cases. Since most of the test cases ran without incident. Re-running of these incident-free test cases on multiple platforms decreases the legacy Test Cases per LOC ratio.

Table 1: Legacy vs. New Project Comparison

	Legacy 7th Iteration	**New 1st Iteration**
FVT Effort	E	0.49 E
Test Cases Run	TC	0.48 TC
Test Cases/Total LOC	TCL	0.36 TCL
Defects/Test Case	DTC	1.8 DTC
Defects/LOC	DFL	0.61 DFL

Because the "new product" was less expansive, only half of the effort was needed for FVT, and only about half as many test-cases were run. Yet, the testing uncovered about twice as many defects per test case. Was the "new code" more defective or were the TDD based test-cases more efficient?

3. Sabre Extreme Programming

In this section we provide information on a case study performed at Sabre Airlines where development teams had been using XP for approximately two years. In this paper, we provide information on the aspects of the case study related to quality comparison.

3.1 Case Study Overview

In a single, longitudinal, holistic [11] case study, we examined a product created by an XP software development team at Sabre Airline Solutions in the United States [6]. We evaluated and compared two releases of the Sabre team's product. In this study, we compared the third and the ninth releases of the Sabre team's product. From this point forth, we refer to the third release as the "old release" and the ninth release as the "new release." The old release was completed just prior to the team's initial adoption of XP; the new release was completed after two years of stabilized XP use. The team used a traditional software process in the old release. Development for the old release began in early 2001 and lasted 18 months. Work on the new release commenced in the third quarter of 2003. In the two and half years that passed from the beginning of the old release to the beginning of the new release, the team became veterans of XP and customized their XP process to be compatible with their environment. This ten-person team develops a scriptable GUI environment for external customers to develop end user software.

Detailed data was collected for each release, and much of this data was gathered from historical resources. The old release was developed approximately two years prior to this study. The researchers were not present for the old release, and the team was not aware that any research would be done on their product or on their documentation. The research team was present only for a portion of the new release development. Many of the necessary metrics were readily available for the new release by examining source code, defect tracking systems, build results, and survey responses.

3.2 Quality Comparison

The case study results demonstrated a quality improvement for the new release in which XP practices were stabilized. Table 2 summarizes quality results which have been normalized to protect proprietary information.

Table 2: Old vs. New Project Comparison

Quality Measures	**Old**	**New**
Internally-Visible Quality (test defects/KLOEC of code)	1.0	0.35
Externally-Visible Quality (released defects/KLOEC of code four months after release)	1.0	0.70

Internally-Visible Quality. Internal (pre-release) defect density, which concerns defects identified by Sabre testers, improved by 65%. Testing was done by the dedicated testers associated with the Sabre team and the developers performing ad-hoc functional testing and unit testing throughout development. We temper these results by noting that these measurements may be skewed because the old release was subject to 18 months of continuous internal testing, while the new release was internally tested for only 3.5 months. Similar to the concerns expressed in the IBM case study in Section 2, the improvement in internally-visible quality might be wholly or partially attributed to a less-thorough testing effort on the part of the new team. Alternately, the fact that the new team wrote extensive TDD test cases throughout development and run them each night (consistent with their use of XP practices) may have made a 3.5 month testing effort equivalent and/or sufficient. The lack of the process suggested by this position paper and its associated metrics prevent an adequate comparison from being made.

Externally-Visible Quality. We observed that the number of defects found in the customer's production system has improved by 30%. The defect numbers presented reflect a collection period of four months after each release. The team's defect rates were below industry averages [5] in both the old and the new releases. Furthermore, no Severity 1 defects were reported for the new release. A Severity 1 defect is classified as a defect that causes the customer's system to be unusable, whereas a Severity 2 defect is a defect where the customer's system is working badly and their operations but a work-around exists for the defect.

Post-release defect counts were impacted by several important factors. One major influencing factor was the doubling of the number of external customers between the old and the new releases. The old release was not used extensively since most customers were awaiting the completion of a new version of the product in progress at that time. However, the new release was used significantly by more customers, some of which had a more complex problem domain than those customers of the old release. Evidence of similar customer use of the product the old and new releases and an assessment of feature complexity would aid in determining the accuracy of the post-release defect comparison for this project. Again, a process for normalizing for these effects would be beneficial.

4. Suggesting a Composite Measure

To adjust for differences in the size and duration of the old release versus the new release, the Putnam productivity parameter (PPP) [8, 9] can be computed. This parameter is a macro measure of the total development environment such that lower parameter values are associated with a lesser degree of tools, skills, method and higher degrees of product complexity. The opposite holds true for higher parameter values [8]. The PPP is calculated via the following equation:

$$PPP = (SLOC)/[(Effort/B)^{1/3} * (Time)^{4/3}]$$

Putnam based this equation on production data from a dozen large software projects [9]. Effort is the staff years of work done on the project. B is a factor that is a function of system size, chosen from a table constructed by Putnam based on the industrial data. SLOC is source lines of code, and Time is number elapsed years of the project.

Perhaps as a community, we can pool results and develop a comparable macro quality parameter which can be used to normalize for effects of testing differences. the number of customers which utilize a product, and defect-removal efficiency (pre-release defects found/total defects found).

5. Summary

In this paper, two industrial case studies were described to demonstrate the complications that can arise when trying to develop theories about whether a certain software development or testing process improves product quality. In both case studies, differences existed which made comparisons indicative but not absolute. This paper suggests that a process be developed for comparing quality data from industrial case studies. Additionally, the community could pool data to create composite measures for comparisons.

Acknowledgements

Many thanks to my research partners on both case studies. On the IBM case study, I worked closely with Michael Maximillien and Mladen Vouk. Lucas Layman and Lynn Cunningham conducted the detailed data collection and analysis for the Sabre Airlines case study.

Biography

Laurie Williams is an Assistant Professor of Computer Science at the North Carolina State University. She received her Bachelor of Science in Industrial Engineering at Lehigh University, her Masters of Business at Duke University, and her Ph.D. in Computer Science from the University of Utah. Dr. Williams worked at IBM for nine years in the Research Triangle Park, NC. During this time, she held various technical and managerial positions in engineering and software development, including three years in a software testing department. She has performed several industrial case studies on software process improvements. Her research interests include software testing and reliability.

References

[1] K. Beck, *Extreme Programming Explained: Embrace Change*. Reading, Massachusetts: Addison-Wesley, 2000.

[2] K. Beck, *Test Driven Development: By Example*: Addison Wesley, 2002.

[3] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison Wesley Longman, 2001.

[4] B. Curtis, "Three Problems Overcome with Behavioral Models of the Software Development Process (Panel)," International Conference on Software Engineering, Pittsburgh, PA, pp.398-399, 1989.

[5] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Boston, MA: Addison Wesley, 2000.

[6] L. Layman, L. Williams, and L. Cunningham, "Exploring Extreme Programming in Context: An Industrial Case Study," to appear Agile Development Conference, Salt Lake City, UT, 2004.

[7] E. M. Maximilien and L. Williams, "Assessing Test-driven Development at IBM," International Conference of Software Engineering, Portland, OR, 2003.

[8] L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, Within Budget*. Englewood Cliffs, NJ: Yourdon Press, 1992.

[9] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 4, pp. 345-361, June 1978.

[10] L. Williams, E. M. Maximilien, and M. Vouk, "Test-Driven Development as a Defect-Reduction Practice," IEEE International Symposium on Software Reliability Engineering, Denver, CO, 2003.

[11] R. K. Yin, *Case Study Research: Design and Methods*, vol. 5, Third ed. Thousand Oaks, CA: Sage Publications, 2003.