Effectiveness of Test Driven Development and Continuous Integration – A Case Study

Chintan Amrit
IEBIS, University of Twente,
Netherlands
c.amrit@utwente.nl

Yoni Meijberg
Topicus, B.V.
Netherlands
yoni.meijberg@topicus.nl

Abstract: *In this article we describe the implementation of hybrid agile practices, namely Test Driven Development (TDD) and Continuous Integration (CI) at a Dutch SME. The quality and productivity outcomes of the case study were compared to a performance baseline set by a reference case, a preceding development project of similar context, size, complexity and team. We observed that on applying TDD and CI, a higher number of defects were discovered compared to the baseline case. The team members at the Dutch SME perceived an increase in the focus on quality and test applications, while considering customer acceptance. As a result of the case study, the Dutch SME now has an infrastructure in place to further evaluate software process improvement (SPI) initiatives.*
Keywords: *Test Driven Development, Continuous Integration, Case Study*

1. Introduction

While most (older) development methodologies show a separate lengthy testing phase at the end of development, TDD comes with a new paradigm, namely, *always* test before coding. Continuous Integration (CI), on the other hand, involves running integration builds and automated tests on the code committed by the developers. Hence CI combines development and testing by enabling unit and functional tests while profiling the application code [1]. CI was designed to improve both the number of testing cycles and the resulting application quality while decreasing the amount of time it takes to find problems and reducing the cost to fix them [1].
Both these practices, however, cost the development company in terms of time and money, as previously suggested by Crosby [2] and later by Dion [3]. Hence, well-grounded empirical evidence is required in order to determine the applicability of these practices in actual industrial settings. In response to this, some empirical studies were conducted on the effectiveness of TDD implementations in academic as well as industrial settings (see Table 1 for a meta review). The outcomes of the empirical studies of TDD implementation seem to indicate an increase in code quality along with an inconclusive effect on developer productivity (Table 1). The extra effort required when writing tests in advance was given as a reason for a decrease in developer productivity. On the other hand, one can argue that TDD could improve internal and external

code quality, leading to lower defect[1] generation and faster fixes, thereby improving productivity. Rafique and Misic [4] mention in their extensive literature review that some experiments with a more detailed design obtained better results on TDD implementation. This was verified in a recent project whereby TDD was implemented successfully [5]. However, this claim has not been tested in an industrial setting adequately and quantitatively nor has it been discussed widely, as noted by Latorre (2013) [5]. Furthermore, most papers do not provide a detailed description of the TDD implementation.

In this paper we describe a case study of both TDD and CI in a Dutch SME. The contributions of this paper are multi-fold. While previous research dealt with few metrics for accessing the cost and quality, we use multiple metrics to access them. Along with an evaluation of the technical quality (also done in previous studies), we propose a quantitative evaluation of the impact of the TDD and CI implementation and provide a detailed account of the case setting. In order to aid future research, we list a set of adherence metrics to measure the extent to which TDD/CI principles have been applied. Our paper also contributes to the growing body of literature on TDD (and CI) implementation, as well as its evaluation in an industrial setting. As in the case with [5], we cannot isolate the effect of the sole application of TDD from the effect of CI.

[1] In this paper we use the terms defect and bug interchangeably.

Side Bar

Literature Overview

We performed a meta analysis of the literature reviews on Test Driven Development published in the last 5 years. In the table below, *internal quality* relates to the quality of the software design (measured by OO metrics, Code density, Cyclomatic complexity etc.), while *external quality* refers to the number of pre/post release defects per given code size, and *productivity* refers to developer productivity (measured using development time, total LOC divided by total effort, hours per feature/development effort per LOC etc.). In Table 1, -NA- refers to the fact that the particular construct was not analyzed.

	Desai et al. (2008)	Turhan et al. (2010)	Causevic et al. (2010)	Kollanus (2011)	Rafique and Misic (2013)
Internal Quality	Little Evidence	Little Evidence	Moderate evidence	Inconclusive	-NA-
External Quality	Moderate Evidence	Moderate Evidence	-NA-	Weak Evidence	Little Evidence
Productivity	Little Evidence	Inconclusive	Evidence for decreased performance	Evidence for decreased performance	Inconclusive

Table 1: Comparison of the findings of literature reviews published in the last 5 years.

Table 1 broadly suggests there is little to moderate evidence that implementation of TDD in an academic or industrial setting is accompanied by an increase in code quality. However, the evidence for an improvement in productivity is largely inconclusive and the literature indicates, to some extent, that TDD implementation is accompanied by decreased productivity.

Desai, C., Janzen, D., & Savage, K. (2008). A survey of evidence for test-driven development in academia. *ACM SIGCSE Bulletin*, *40*(2), 97-101.
Turhan, B., Layman, L., Diep, M., Erdogmus, H., & Shull, F. (2010). How Effective is Test-Driven Development. *Making Software: What Really Works, and Why We Believe It*, 207-217.
Causevic, A., Sundmark, D., & Punnekkat, S. (2011, March). Factors limiting industrial adoption of test driven development: A systematic review. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on* (pp. 337-346). IEEE.
Kollanus, S. (2011). Critical issues on test-driven development. In *Product-Focused Software Process Improvement* (pp. 322-336). Springer Berlin Heidelberg.
Rafique, Y., & Misic, V. B. (2013). The effects of test-driven development on external quality and productivity: a meta-analysis. *Software Engineering, IEEE Transactions on*, *39*(6), 835-856.

2. The Case Context

The data for this research originates from a two-case comparison of software implementation at a Dutch SME. The two cases were consecutive software development projects from within a single software house seated in the Netherlands. The first case followed a development process with neither any methodology alterations, nor any involvement by the researchers, while the second project followed a process refined by TDD and CI principles. The second author was present as

both an information analyst and chief methodology implementer of the second project. We use the first project as a reference case to compare the potential differences in the outcome of the two projects.

Both the software development projects in the Dutch SME dealt with healthcare claim handling. Healthcare claims are mostly digital transactions (involving the transfer of messages, digital invoices or money) between healthcare providers and insurers. As soon as a person receives some form of healthcare by a provider (for instance surgery), the provider usually makes several transactions that need to be appraised by the insurer. On average, each person in the Netherlands is involved in about ten transactions per year. This implies that managing these digital transactions is a high volume industry. As these transactions are highly standardized by the Dutch government, they are well suited for automation. The Dutch SME of this case study makes software applications that provide such automation. The software project that we describe in this article is based on one such software application. For the sake of clarity, we denote the software development project in which TDD and CI testing was implemented as Case Study 2(CS2), and the previous version where it was not, as Case Study 1(CS1).
The CS1 project served mainly as an entry portal for incoming healthcare transactions from healthcare providers, at an insurance intermediary. Its main goal was to structure, check and help in the appraisal of incoming transactions. After this process, the transaction was entered into the financial database.
On the other hand, the CS2 project's software application was installed at an invoicing intermediary. An invoicing intermediary takes care of everything around healthcare transactions for a set of healthcare providers. This is done so that the healthcare providers can concentrate on providing proper care, instead of getting bogged down with the many financial transactions that are required. However, in terms of functionality, the CS1 and the CS2 projects were very similar.
To summarize, the applications of the two cases were installed at different points of the healthcare chain, and thus satisfied different stakeholders. They were similar in terms of functionality, as they interacted (indirectly) within the same industry while applying common transaction standards. The CS2 implementation was, in particular, based on the CS1 architecture and most of the CS2 modules were refactored CS1 modules. As 92.5% of the modules overlapped between the two architectures, we can safely assume that the number and complexity of the features were similar. This is also more than the 75% overlap reported by Latorre (2013)[5].

Table 2 lists the similar project characteristics found in the two cases or development projects in terms of context factors and software product measure factors. The factor list is taken from the work in [6-8]. We notice that both kinds of factors, especially the contextual factors are comparable and rather similar between the two case studies.

Characteristics	Case Study 1 (CS1)	Case Study 2 (CS2)
Context Factors		
Application	Transactional system for technical verification	Transactional system for technical

	and appraisal, administrative handling and payment of healthcare declarations	verification, routing and appraisal of healthcare declarations
Customer	Authorized insurance intermediary	Invoicing intermediary
Duration (time)	8 months	10 months
Average monthly effort (man-days)	78	73
Total effort (man-days)	732	731
Team size	<10	<10
Team people overlap	-	>4
Experience level (<5 years, 6-10 years, >10 years)	Most members < 5 years experience	Most members < 5 years experience
Project manager's expertise	> 5 years experience	> 5 years experience
Applied technology	C# ASP.NET MSSQL	C# ASP.NET MSSQL NHibernate
Product Measures		
Source LoC	28049	21340
Maintainability Index Average	79.20	85.32
Cyclomatic Complexity Average	280.53	297.71
Depth of Inheritance average	2.62	4.70
Class coupling average	57.59	77.65

Table 2: Description of the context and product measures of case study 1 and 2

3. Measurement Metrics

In order to measure the effectiveness of TDD and CI, we considered three perspectives, namely,

1. Defect Reduction: Whether TDD and CI helped in reducing the number of defects, i.e., if CS2 has fewer defects compared to CS1
    a. Pre- release defect level – the number of defects detected in the pre- release software per KLOC [9]
    b. Post- release defect level – the number of defects occurring in the post- release software per KLOC [9, 10]
2. Defect lead and throughput: Whether TDD and CI helped in reducing the time to find and fix the defects
    a. Defect resolution duration – the duration between the discovery and closure of defects, given by:
    $\frac{\sum_D d_{closedate} - d_{reportdate}}{|D|}$, where |D| is the number of defects

    b. Defect pre/post solvability – the proportion of defects uncovered prior to and after the release given by $\frac{|D^{Pre}|}{|D|}$, per pre- solvability release, and $\frac{|D^{Post}|}{|D|}$ per post-solvability release

3. Development Productivity: The development productivity and rework rate in development per release, given by: $\frac{KLOC}{Effort}$, where the KLOC is the amount of KLOC added per release.

4. Results

We first analyzed our data using descriptive statistics and then we considered inferential statistics. 39 versions of the CS1 software and 24 versions of the CS2 software were released in the period from the start-up of the two projects up to when the data for this study was collected. Both projects used the Mantis bug tracker, and the developers and the customers submitted the bug reports. We analyzed the CS1 and CS2 Mantis bug trackers and removed duplicates and issues that were not really bugs. CS1 had 414 viable bug reports for analysis while CS2 had 474 viable bug reports. The bugs were prioritized in both the CS1 and CS2 projects, in terms of defect severity. But this was not done explicitly in both projects. The developers and project leaders knew which bugs to resolve first, based on which part of the code they occurred in, and this knowledge was tacit (not made explicit in the bug tracker). In the case of CS1, the prioritization and resolution was done manually. In both projects the bugs that caused build failures were fixed first and the rest were fixed in the order of prioritization. In the case of CS2, the Continuous Integration (CI) helped in this regard, as the CI -related best practice of "executing all tests with every build and making a single failed test fail the build" [1] was followed.
When we asked the project managers of both projects to compare the list of bugs (in terms of the number of high priority bugs and the severity of the defect) in both projects, they both agreed that the bugs from both the projects were comparable.

Descriptive Statistics

The box plot of the distribution of pre- and post- defect resolution data is shown in Figures 1a and 1b; the numbering of the figures that follow is based on this categorization. In Fig. 1a we notice that the defect level of CS2 is consistently higher (before the release) than the defect level of CS1. This could be the result of implementing the test first approach in the CS2 TDD, and not necessarily because the CS1 software had fewer bugs.
This seems to be verified by Fig. 1b where we notice that CS1 has significantly more bugs than CS2 after the release. Figure 1b also demonstrates that the medians of the post-release number of bugs per KLOC are nearly the same for CS1 and CS2. This shows that the CS2 project had a more consistent number of bugs per release, and fewer releases with a large number of bugs (as was the case with the CS1 project).
In Figures 1c and 1d we can see the results of the defect lead and throughput category metrics. Fig. 1c indicates that the Defect resolution duration (the time in days between discovery and resolution) is mostly higher for CS1 than CS2 (although the medians are nearly the same). Furthermore, CS1 has a much longer tail indicating that a few bugs required more fixing time compared to the CS2 project.

In Fig. 1d we see the Defect pre/post solvability (the proportion of defects uncovered before release). The CS2 project again outperforms the CS1 project because a consistently higher percentage of bugs were found before a major release. The developers in CS2 also accurately flagged a bug as solved as soon as they had fixed a defect. This was done to prevent colleagues from fixing the same defect, and to show the progress to the customer. Hence, this data can be considered reliable.
In Fig. 1e we see the comparison of development productivity between the two projects. The inter-quartile range of the CS1 project is almost double to that of the CS2 project, while the medians are nearly the same. This implies that while the development productivity was nearly the same, there were instances when the number of added KLOCs was very large for the same number of man-days. When we approached the team members for an explanation, they thought that it could be due to the introduction of large pre-coded components (containing several hundred code-lines apiece).

Figure 1: The descriptive statistics for the metrics described in Section 3

Inferential Statistics

We first tried to ascertain the difference in means between the CS1 and the CS2 project, with respect to the metrics listed in Section 3. The common statistical method used to test the difference in the distributions of two samples is the Student's T-test. However, the prerequisites for using the Student T-test are that the two distributions must have the same variance and both populations should follow the normal distribution. We could not use the Student's T-test as Levene's test showed that some of the metrics from the CS1 and the CS2 projects did not have equal variances, while the Shapiro-Wilk's test for normality demonstrated that normality was violated by almost all the metrics. The reason behind this was the variation in the denominator in all the metrics, namely KLOC. When comparing the KLOC between CS1 and CS2, the Levene's test showed a significance of 0.38 (indicating that the population variances are equal) and Shapiro-Wilk's showed a significance of 0.0 for both projects (indicating that it is not normally distributed). As the distribution was not normal for both the samples, we considered a non-parametric test, namely the Man-Whitney Wilcoxon (MWW) test.

Metric	Mann - Whitney U	Wilcoxon W	Z	Asymptotic Significance (2-tailed)
Pre-release defect level	258	1038	-.62	0.009
Post-release defect level	388	619	-.33	0.739
Defect resolution duration	55555.5	154790.5	-2.57	0.010
Defect	312	942	-1.84	0.067

pre/post solvability				
Development productivity	413	713	-.78	0.436

Table 3: Mann-Whitney test's results for the metrics in this paper

Table 3 groups the results based on the relevant metric category (Section 3). The p-value of the Asymptotic Significance of the Pre-release defect level and Defect resolution duration is less than 0.05. Hence, these two metrics demonstrate the CS2 project outperforms the CS1 project. Regarding the Defect pre/post solvability metric, the p value is just over 0.05 whereas for the Post- release defect level and Development productivity metrics, we find no significant difference between the CS1 and the CS2 project. It has to be noted that the Mann-Whitney Wilcoxon test can be used to only determine if the two distributions are indeed significantly different, and not which distribution has a higher median (and other measures of dispersion). Therefore, we combined the data from Table 3 with Figures 1a and 2a. We see that the number of pre- release defects found in CS2 is significantly greater than CS1 (Fig. 1a). We also see that the *Defect resolution duration* of project CS1 is significantly larger than that of project CS2. Though the U value of the *Defect resolution duration* appears to be rather large, it is more-or-less what we can expect given the large number of samples, 414 for CS1 and 474 for CS2 (an expected estimator of U is multiplying half of the sample sizes of the distributions). In the case of *Defect pre/post solvability* we see that the MWW test p value just exceeds 0.05, though Fig. 2b demonstrates that the number of defects uncovered before release is much larger in CS2 than CS1. Hence project CS2 outperforms CS1 in this metric. With respect to the *Post- release defect level,* the MWW test shows that the distribution of the number of bugs per KLOC of the two projects is not significantly different. We see from Fig. 1b that the medians of the two distributions are indeed quite close, though the CS1 upper quartile is much larger than that of CS2, and so, there were more defects occurring in CS1, post- release. This is also the case with the *Development productivity* metric; we notice from Fig. 3 the medians are similar but the lower and upper quartiles are largely different, implying that the development effort put into a few releases in CS1 was much larger than in CS2.

Discussion and Conclusion

In this paper we analyzed the effectiveness of implementing TDD and CI, through a case study. The team members at the Dutch SME did perceive an increase in the focus on quality and in the application of tests, whilst considering customer acceptance. The company now has an infrastructure in place to further evaluate other software process improvement (SPI) initiatives.

Though the inferential statistics are not conclusively in favor of the TDD and CI case (CS2), the descriptive statistics point to an overall improvement in not only finding more defects (Defect Reduction), but also in shortening the time required to fix the defects (Defect lead and throughput). One of the limitations of this paper could be the usage of KLOC in most of the metrics (in the denominator), as well as the usage of KLOC for measuring development productivity (Fig. 3) [11]. Shihab et al. (2013) [11] mention that metrics like Cyclomatic

Complexity, and a combination of software metrics, outperform LOC in estimating effort, whereby using solely LOC underestimates effort (see Shihab et al. (2013) [11], page 1991). However, if the same metric is used to compare different projects, the underestimation of effort could balance out - as it is underestimated for both cases, and we are only interested in the comparison of development effort and not the exact development effort. Yet, applying KLOC limited our usage of different statistical techniques, as the variance it introduced made the distributions non-normal.

In summary, the contributions of this research are multiple:

1. We have added to the small but hopefully growing body of empirical literature on agile implementation in an industrial setting, such as [12-14]
2. We have endeavored to provide a rich description of the project setting (contextual and product in Table 2) that we think is useful to recognize potential validation errors.
3. We have added to the existing set of metrics (Section 3) and we think our new metrics give a richer and more detailed description of the effects of Agile implementations in general.

References

[1] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*: Pearson Education, 2007.

[2] P. B. Crosby, *Quality without tears: the art of Hassle-Free management*. New York: McGraw-Hill, 1984.

[3] R. Dion, "Elements of a process-improvement program [software quality]," *Software, IEEE,* vol. 9, pp. 83-85, 1992.

[4] Y. Rafique and V. B. Misic, "The effects of test-driven development on external quality and productivity: a meta-analysis," *Software Engineering, IEEE Transactions on,* vol. 39, pp. 835-856, 2013.

[5] R. Latorre, "A successful application of a Test-Driven Development strategy in the industrial environment," *Empirical Software Engineering,* pp. 1-21, 2013.

[6] T. Bhat and N. Nagappan, "Evaluating the efficacy of test-driven development: Industrial case studies," in *ISCE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering*, 2006, pp. 356-363.

[7] A. D. Carleton, R. E. Park, W. B. Goethert, W. A. Florac, E. K. Bailey, and S. L. Pfleeger, "Software Measurement for DoD Systems: Recommendations for Initial Core Measures," Software Engineering Institute, Pittsburgh, PA1992.

[8] N. Nagappan, E. M. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: Results and experiences of four industrial teams," *Empirical Software Engineering,* vol. 13, pp. 289-302, 2008.

[9] UKSMA, "Defect Measurement Manual," United Kingdom Software Metrics Association, Edenbridge, United Kingdom2000.

[10] N. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. London: International Thomson Computer Press, 1996.

[11] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "Is lines of code a good measure of effort in effort-aware models?," *Information and Software Technology,* vol. 55, pp. 1981-1993, 2013.
[12] C. Amrit and J. van Hillegersberg, "Detecting Coordination Problems in Collaborative Software Development Environments," *Information Systems Management,* vol. 25, pp. 57 - 70, 2008.
[13] M. Daneva, E. Van Der Veen, C. Amrit, S. Ghaisas, K. Sikkel, R. Kumar*, et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *Journal of systems and software,* vol. 86, pp. 1333-1353, 2013.
[14] C. Amrit, J. van Hillegersberg, and K. Kumar, "Identifying Coordination Problems in Software Development: Finding Mismatches between Software and Project Team Structures," *arXiv preprint arXiv:1201.4142,* 2012.

Author Bios

Chintan Amrit is an assistant professor at the department of Industrial Engineering and Business Information Systems (IEBIS), at the University of Twente. He holds a master's degree in Computer Science from Indian Institute of Science, Bangalore and a PhD in Information Systems from University of Twente. In the past he has worked for a period of three years for a biometric software company in Germany. His research interests are in the area of software development, business intelligence (using machine learning), data analysis methods, supply chain logistics and mining software repositories. He is the Coordinating Editor of Information Systems Frontiers and a regular track chair at ECIS conference. He can be reached at c.amrit@utwente.nl

Yoni Meijberg is a managerial change agent at Topicus in the Netherlands. His interests lie in areas software project management, DevOps, Agile, continuous delivery, Lean and business research methods. He has a master's degree from RSM Erasmus University on Organizational Change and one from University of Twente in Industrial Engineering. He can be reached at yoni.meijberg@topicus.nl