Impact of Continuous Integration on Software Quality and Productivity

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Arka Bhattacharya

Graduate Program in Computer Science and Engineering

The Ohio State University

2014

Master's Examination Committee:

Dr. Jay Ramanathan, Thesis Advisor

Dr. Rajiv Ramnath
Abstract

It is widely accepted that a defect detected in the earlier stages of the Software Development Lifecycle is less expensive to fix than one found in the later ones. Also that the practice of Continuous Integration (CI) facilitates this early detection and rectification of defects through continuously integrating and testing code with periodic builds each time new code is checked into version control. However, evidence of CI benefits is primarily anecdotal, even though this practice has been around a while.

This research provides empirical evidence for the claims of benefits from CI based on the correlation of CI practice maturity among application development teams and improvement in software quality and productivity over time using metric data reported within a Fortune 100 enterprise. Gathering quality and productivity data on a macro and micro level within the enterprise we have tried to support the CI claims. These are 1) increase in the defects getting detected in pre-production and 2) improvement in productivity among development teams. Finally, we provide an insight into better ways for quality data collection in addition to the lessons learnt through the research.
Dedication

This thesis is dedicated to my family.
Acknowledgments

I owe my deepest gratitude to Dr. Jayashree Ramanathan and Dr. Rajiv Ramnath for providing me with the opportunity for this research. Thank you Prof. Jay for being so tolerant when I sent numerous revisions of my draft your way. I cannot be grateful enough to Prof. Rajiv for the internship opportunity that set the ball rolling for this study. I would like to immensely thank Carmen DeArdo for his constant guidance while I was carrying out my research in the industry. Those countless hours spent brainstorming were tremendously beneficial. Sam Patino, you sir have been very crucial to the process of data collection in one of the major areas in this study. I thank David Ziels for taking time off of his busy schedule to hear my ideas out and provide me with the necessary data for the research. Eapen Thomas, I owe you a lot for your deep insights that has been extremely beneficial for conducting this study.

My friends Satyajeet, Piyush, Siddharth, Pallavi, Abhishek Mishra, Abhishek Dharmapurikar and numerous other friends in OSU who have been through thick and thin with me in the journey called Graduate Studies. My surrogate mother in the US, Julia Morris Groves who has given me the necessary jolt every now and then when the motivation in me faded.

My Mom, Dad, brother, my grandparents who believed in me even when I had lost faith in myself.
I thank you one and all for being there for me and having contributed towards my work in ways one can never imagine.
Vita

2003 to 2005 ................................Alfa Junior College, Visakhapatnam,
Andhra Pradesh, India

2005 to 2009 ..................................B. Technology, Computer Science, West
Bengal University of Technology, India

December 2009 to July 2012 ......................Systems Engineer, Tata Consultancy
Services Ltd., Calcutta, India

March 2013 to August 2014 ......................Intern, Build-Capability, Nationwide
Insurance, OH, USA

August 2014 to present ..........................Graduate Student, Department of Computer
Science & Engineering, The Ohio State
University, OH, USA

Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ..................................................................................................................... ii
Dedication ................................................................................................................... iii
Acknowledgments ...................................................................................................... iv
Vita ............................................................................................................................. vi
List of Figures ............................................................................................................. ix
Chapter 1: Introduction ............................................................................................. 1
  1.1 Motivations for further CI study ........................................................................ 1
  1.2 Approach ........................................................................................................... 4
  1.3 Validation .......................................................................................................... 6
  1.4 Contributions .................................................................................................... 7
  1.5 Organization of this thesis ................................................................................ 7
Chapter 2: Related Work .......................................................................................... 8
Chapter 3: Limitations and Challenges in Data Procurement and Analysis ............. 11
  3.1 Data Set Size Limits ......................................................................................... 11
  3.2 Quality Data Limitations .................................................................................. 12
  3.3 Productivity Analysis Limitations ..................................................................... 14
Chapter 4: Continuous Integration Framework ......................................................... 15
Chapter 5: Quality and Productivity Metric Analysis .............................................. 20
  5.1 Validation Data ................................................................................................ 21
    5.1.1 Data for Quality Analysis ........................................................................ 21
    5.1.2 Data for Productivity Analysis ................................................................. 22
  5.2 Analysis ............................................................................................................ 23
    5.2.1 Quality Metric Analysis .......................................................................... 23
    5.2.2 Productivity Metric Analysis .................................................................. 34
    5.2.3 DC Quality Data Analysis ....................................................................... 43
Chapter 6: Conclusion ............................................................................................. 47
References ................................................................................................................. 50
Appendix A: Quality Data

53
List of Figures

**Figure 1** Illustrates the factor of increase in expenses to fix defect in SDLC.................. 2

**Figure 2** DC & ITSA matrixed relationship................................................................. 5

**Figure 3** The CI architecture in the enterprise ............................................................ 16

**Figure 4** Phases in SDLC as observed in the enterprise ............................................... 24

**Figure 5** ITSA quality data and control charts............................................................ 30

**Figure 6** Top: Control Chart differences **Bottom**: Quality trend differences.............. 31

**Figure 7** PI data for DC projects for 2011 .................................................................. 36

**Figure 8** PI data for DC projects for 2012 .................................................................. 38

**Figure 9** PI data for DC projects for 2013 ................................................................. 40

**Figure 10** Overall trend in PI data for all years............................................................ 41

**Figure 11** Production defect percentage found in the enterprise from Jan-June 2014..... 44

**Figure 12** Production defect percentage by DC from Jan-June 2014 ......................... 45

**Figure 13** App-1 raw quality from 2011-2013............................................................. 53

**Figure 14** App-1 quality trend...................................................................................... 54

**Figure 15** App-2 raw quality data 2011-2013............................................................. 55

**Figure 16** App-2 quality trend...................................................................................... 56

**Figure 17** App-1, App-2 consolidated raw quality data.............................................. 57
Figure 18 App-1, App-2 consolidated quality trend .......................... 58
Chapter 1: Introduction

Since software development is a means for introducing change into a business and operations is the activity concerned with providing stability to the current state of business, there always exists a discord between the two [1]. This friction has put the need for collaboration between the two activities into prime focus. This is addressed by DevOps [1], which is a new software development process model that blurs the line of clear distinction between Development and Operations and tries embracing change by not considering it to be a deterrent to the establishment of stability. Continuous Integration (CI) is one such practice that falls under the DevOps umbrella.

1.1 Motivations for further CI study

CI as a process involves daily integration and testing of developer code to eliminate defects while the application is pushed through the stages of Software Development Life Cycle (SDLC) and into production. As the practice suggests, it is designed to enable an earlier detection of bugs within the initial stages of software development rather than the latter ones. The objectives of this process are to enhance the quality of the application being developed and also make the rectification procedure of encountered defects less expensive. As stated by Barry Boehm, the earlier the defects are detected the less expensive their
resolution. This is also the most important philosophy that drives CI. **Figure 1** below gives a fair estimation of the increase in expenses involved in fixing defects as they are caught in the subsequent stages of the Software Development Life Cycle (SDLC).

![Figure 1](image)

**Figure 1** Illustrates the factor of increase in expenses to fix defect in SDLC

In addition to improving software quality, the CI practice is also attributed with a reduction in the risk to delivery as integration errors and build related issues are detected during early phases such as development [2]. By providing continuous feedback after every cycle and
a reduction in repetitive manual processes, CI enhances programmer productivity by freeing developers to do more thought provoking work [3]. The practice of CI requires invariance in the process cycle in each run to guarantee consistency.

However, most of the existing research on the benefits and impact of CI is anecdotal or survey-based. This forms the main motivation for this case study based research into the impact of CI within the enterprise. The thesis presents data-driven evidence to support the main hypothesis: *Continuous Integration (CI) increases the quality of applications developed by detecting defects early in the SDLC along with development productivity.*

The approach is to establish the above hypothesis with four main claims using enterprise data. These claims are as follows:

**Claim 1:** CI detects defects earlier in the SDLC,

**Claim 2:** CI improves software quality in production,

**Claim 3:** CI improves productivity, and

**Claim 4:** Within the enterprise the improvements of CI are more pronounced at a macro level and not always detectable at the micro project level. However, using profiles to
understand micro-level variations we find that there is an overall positive contribution from the micro to the macro-level providing support for Claims 1-3.

1.2 Approach

The research is fuelled by metric data from two macro-level areas of a large insurance enterprise. The organizational structure of the enterprise plays a crucial role in the way data has been collected for the analysis. The enterprise performs its business through software applications that are grouped by similar business functions and all applications catering to one common business need fall under one IT Solution Area (ITSA). There are multiple ITSAs that are governed by the office of the Chief Information Officer (CIO). Each of the applications within an ITSA have an application development team that caters to the development needs of that application. For the purposes of this study we focus on the JAVA/J2EE application development teams within an ITSA that use Hudson [4] as the tool for practicing CI. In addition to an ITSA, metric data from the Development Center (DC) is also used in the analysis. The DC is at the same level as an ITSA in the organizational echelon, which consists of multiple application development teams that service different ITSA needs. Figure 2 illustrates the relationship between an ITSA and the DC.
For verifying Claims 1 and 2 quality metric data is obtained for application development teams practicing CI from an ITSA and the DC. It is then analyzed to observe a pattern that suggests an improvement in software quality of the deliverables being released by those application development teams as they keep practicing CI. Claim 3 surrounding productivity of development teams is verified by collecting metric data for productivity. Productivity is measured in terms of Productivity Index (PI), which is calculated through the SLIM (Software Lifecycle Management) [5] tool provided by an organization called Quantitative Software Management (QSM) [6]. The tool takes project specific information to evaluate the PI [7]. PI is calculated for projects that have been developed using CI in the
enterprise. The data visualization features provided by SLIM help plot the PI values of projects relative to the rest of the industry. This gives an idea of how the productivity of projects in the enterprise compares with the industry. The data for both quality and productivity metric collected at the ITSA and DC levels are jointly analyzed to provide support for **Claim 4**.

### 1.3 Validation

This is based on metric data for quality and productivity that has been gathered for a period of 3 years which is the duration of CI practice within the enterprise. The method of validation adopted for **Claims 1 and 2** is by extracting defect numbers for 6-month periods to capture the behavior of CI with the variation in the nature of development activity and determining if the percentage of defects found in production in those periods decreases with the progression of time. A steady decrease in the percentage of production defects indicates that applications are rolled out with less defects. In order to support **Claim 3** which is an improvement in productivity we track the Productivity Index (PI) data for completed projects out of the Development Center (which is also considered as an IT Solution Area) and observe if the mean, median, minimum and maximum PI increases with time after the introduction of CI in the organization. As we carry out analysis for **Claims 1, 2 and 3** for data at the ITSA and DC levels the existence of **Claim 4** becomes evident.
1.4 Contributions

- Established that CI detects defects early on in the SDLC by showing less defects are released into production which also in turn increases software quality, the extent of which is dependent on the kind of development work the application is undergoing.
- Developer productivity gets improved through the usage of CI as PI values for projects overall increase with time as the application development teams continue to follow the practice.
- The improvements in quality and productivity due to CI are shown to be more pronounced at the macro level through data gathered at the ITSA and DC level.

1.5 Organization of this thesis

We begin by concisely describing previous work done related to CI in Chapter 2 and how our current research is an addition to the existing research. Following which we describe the challenges encountered in gathering the metric data within the organization in Chapter 3. Chapter 4 describes the architecture of the CI environment used in the enterprise and the technical specifications that conforms to Martin Fowler’s pre-requisites to an ideal CI environment. We present our analysis in Chapter 5 and the conclusions in Chapter 6.
Chapter 2: Related Work

The practice of CI from the perspective of developing a feature to an already existing application has been described by Martin Fowler [8]. In this work the architecture required for setting up the CI environment and the movement of code from version control to deployment in the cloned production environment is discussed [8]. The paper also mentions that CI leads to easier bug detection and rectification [8]. CI techniques with Test Driven Development have been applied to embedded system software development [9]. The benefit of CI in embedded system has been defined as the ability to pinpoint the origin of an unexpected failure [9]. As a result of this bugs are easier to detect and rectify thereby increasing the life of the system [9]. The practice enables a smoother means to add features to already existing software [9]. Automation of processes comprising CI enable rapid software development and lessen the requirement for greater debugging [10]. This is concluded by Geiss in his work with Jenkins as the CI tool while applying CI techniques to android mobile application development. The research based on Fowler’s work also provides information regarding the subsystems of the CI architecture like build tools, version controls and CI tools available for different development platforms. The other benefit of CI speeding up software delivery by reducing integration times is discussed by Stolberg [11] and also by Hembrink [12]. This paper also validates the ten sub-practices
that comprise CI as listed by Fowler’s work by implementing it for developing Windows .Net C# applications. Moreover, the work emphasizes the compatibility of CI with the agile method of software development. The quality of the process of CI and the effect the discipline of developers in checking-in code has on it is discussed by Modesto [13]. The study claims that the probability of the build breaking increases with more developers checking-in code. Also an increase in the length of the feedback cycle is observed as with the increase in code size the build time is longer. Stahl & Bosch [3] obtained quantified data through interviews from four product development projects in Ericsson to conclude that CI improves communication within teams and also leads to an increase in developer productivity. Recent work [2] elucidates the reason behind the unevenness in the benefits observed by various teams practicing CI. It documents the differences in the implementation of CI among teams on a case basis.

This thesis complements previous work done on Continuous Integration by providing validation for the claims that the practice improves quality of software developed and enhances programmer productivity using metric data right out of the industry. Similar to the earlier work by Geiss on using Jenkins as the CI tool for Android application development this study is based on examining metric data from JAVA/J2EE web application development, which use Apache Hudson as their tool for CI purposes.
Motivation

Effectiveness of a practice that is to be introduced in the industry is better examined using real life industry data post-deployment. Data from an enterprise contains a lot of variance whose existence is valid in the real world scenario as opposed to a controlled experiment where the parameters can be tuned in accordance with the study.

In need of a more pragmatic way to verify the stated advantages of CI this thesis quantifies the performance of the practice using Quality and Productivity metrics. Drawing quality and productivity data from several application development teams operating in an enterprise this study provides empirical evidence to validate the claims of advantages of CI. Like Geiss’s work on Jenkins this study determines the behavior of metric data as agile lines use Hudson as their CI tool for development. Similar to earlier work where CI is used to develop embedded system software, android mobile applications and applications on the .Net framework, the focus here is the response of the development of web applications based on JAVA/J2EE.
Chapter 3: Limitations and Challenges in Data Procurement and Analysis

3.1 Data Set Size Limits

An enterprise whose functioning has a strong dependency on software applications developed using different technologies does contain a significant number of application development teams. The existence of numerous application development teams makes the introduction of a software development standard across the whole enterprise cumbersome. Also, response to the introduction of a new practice across all teams is guaranteed to be varied depending on the team’s ability to cope to a change. This is dependent on the learning ability of the developers constituting that team and also the criticality of the application that is being developed. If a mission critical application has a constant need for development then the introduction of a new standard in development that involves a higher learning curve would be viewed as undesirable. Applying these causes to the introduction of CI in the enterprise, this study limits the number of application development teams to only one ITSA. Also the organization in perspective has CI capability for only JAVA/J2EE applications. Furthermore, the varying maturity levels among CI practicing application
development teams places a constraint in the data set size as only the ones with the longest maturity period of three years were appropriate for the study.

3.2 Quality Data Limitations

The focus is on one ITSA that has been following CI practices right since the introduction of CI in the enterprise around the second half of 2011. This ITSA has 20 JAVA/J2EE application development teams and due to the absence of a record documenting the ones following CI it was harder to filter out the non-CI application teams. However, the low number of non-CI following application teams has been expected to introduce minimal noise in the quality data gathered for the whole ITSA.

The study on software quality is dependent on data extracted out of the defect repository which has the capacity of storing only the data corresponding to a 3-year period. Absence of any data prior to 2011 makes it harder to make any before and after sort of assessment of an application development team’s response to CI.

In order to provide a finer trend analysis the duration of data gathering is broken down to a 6-month period. This breakdown of the duration of analysis reduces the data size as there exists an uneven distribution in the development work. Since, the functions of an insurance organization conforms to a fiscal year there is a tendency of the development work to be
more focused towards producing deliverables in one second half of the calendar year and to maintenance and bug fixes in the other. As a result of which there exists an uneven trend in the quality data with time.

The utilization and logging of defects in the defect repository has grown over the years among development teams resulting in the appearance of more defects numbers in the later parts of the duration in our analysis.

As described in the previous work the CI tool helps pin-point the source of a build break or test fail to the development team and sends this feedback to the application development team [12]. If the causes of each of these breaks in the CI cycle were determined and archived in the defect repository then its inclusion in the analysis would have acted as a better validator for the claim that more bugs are detected earlier in the SDLC.

Since, a defect archived in the quality repository could be due to infrastructural issue, a requirements issue or a design issue only the defects arising out of a fault in the application are included for the purposes of this study. In addition to the aforementioned defects, sometimes testers log defects, which are duplicates to defects pre-existing in the repository, or are found out to not be a defect after re-examination. Filtering out such defects also drastically reduce the data set size while collecting quality data on an application level.
3.3 Productivity Analysis Limitations

Adherence to the practice once introduced and reporting of metrics that measure its performance also differs across teams. This forces a shift of focus to a different area called the Development Center (DC) for the productivity data analysis. The reason being that the method adopted by DC in reporting project data that serves as input to the SLIM tool results in accurate calculation of the Productivity Index (PI) metric. This is not the case with the ITSA chosen for quality analysis where the project data is reported on a monthly basis and not on the basis of the project duration that results in skewed calculations for PI. Analyzing the behavior of one ITSA under the influence of CI on the basis of quality and productivity data both would have provided the basis for establishment of a causal relationship. However, it would suffice to say that apart from the dynamism in the application development teams there isn’t much difference between DC and the ITSA considered for quality study. Both these areas started using CI techniques around the same time and the mixture of developers with varying experience levels working in both the areas are similar. The focus of application development teams in the ITSA do not differ application wise whereas the application development teams in the DC do not cater to one specific application for over a long period of time.
Chapter 4: Continuous Integration Framework

The basic structure and functioning of the CI framework is exactly similar to the one described by Fowler. CI environment setup in the organization in perspective is illustrated in Figure 3 below. The Software Configuration Management used to host developer code here is the Apache Subversion denoted by SVN [14]. The CI server used in this context is the Apache Hudson [4] which is a plug-in based application that can be configured from its Management Console to check code out on a periodic basis, compile and build the code using Apache Maven [15]. Once the build completes the code can be deployed onto a JAVA/J2EE Application Server, IBM Websphere Application Server 8[16] is the one preferred as part of CI practice in the organization in consideration. Hudson being a plug-in based application connects with external entities like Subversion and Application Servers via plug-ins that could be installed through the management console. Once the code is deployed automation scripts in cucumber [17] are run for performing automation testing. Hudson is also linked to SONAR [18] that provides the capability of doing code quality analysis so that the code being developed adheres to the standards set by the organization.
The tool is configured to poll for code changes checked in by developers. Every check-in results in a complete build of the mainline of the code and any errors encountered in the build is considered as an integration error and a feedback is sent to the programmer tagged by the check-in of the code causing the failure. The compile, unit test, build, deploy and automation test cycle is setup in Hudson through jobs. The termination of a job signifies the completion of a cycle. The reaction time to a build or a test fail is a major contributor to the software quality of the application being developed in the process [2]. At times of

Figure 3 The CI architecture in the enterprise
intensive development there results in multiple builds per day ensuring a better detection of defects in the development stage of the Software Development Life Cycle.

Adhering to Martin Fowler’s list of requirements to ensure that CI best practices are followed we provide the full list of specifications that satisfy each criteria.

**Maintaining Single Source Repository:** Development teams practicing CI check their code in to Apache SVN, the standard version control in the enterprise. Each developer checks in code to the mainline that exists in the repository that acts as the source for multiple builds.

**Automate Build:** Hudson, the CI tool is configured to build code by polling the code repository each time a user commits to the mainline. The server hosting Hudson has Maven configured to make builds. Notification is sent to the user through Hudson each time post-build completion.

**Make your Builds Self-Testing:** The agile development practice in the enterprise follows Acceptance Test Driven Development [19]. After each build is deployed on a test environment Cucumber scripts are run to test the functionality of the application.

**Everyone Commits to the Mainline Everyday:** A development team usually has more than one developer writing code to different modules of a project. Each developer commits his
code to the mainline pertaining to the entire project daily either once or more depending on the amount of code written after performing unit-tests on it on his local machine.

*Every Commit Should Build the Mainline on a Separate Integration Machine:* As mentioned earlier Hudson polls the code repository for updates. Each update results in a build of the complete project code of all developers.

*Keep the Build Fast:* The build is always under a few minutes unless the code base is huge which takes around 10 minutes almost to complete the build.

*Test in a Clone of the Production Environment:* Since the focus is J2EE applications, IBM Websphere Application Server 8.0 is the choice of the enterprise for an application server. The test and production application servers are the same version to eliminate any defects caused by infrastructure mismatch.

*Make it Easy for Anyone to Get the Latest Executable:* The source code for J2EE applications are compiled and built into an Enterprise Archived Resource, which is commonly known as an EAR file that is then deployed on an application server. Each build makes this EAR file easily available.

*Everyone Can See What’s Happening:* After each build an email is sent informing developers of the build status. Thus, developers can fix any build related issues in their own code, post-build.
Automate Deployment: Hudson has a plug-in that allows it to automatically deploy an EAR file on to the application server. This plug-in was developed in-house for the enterprise and meets Fowler’s deployment automation criteria.
Chapter 5: Quality and Productivity Metric Analysis

Discussed earlier, the two metrics used in validating the above claims are quality and productivity.

**Quality** from a manufacturing perspective is defined as the conformance to design and requirement specifications [20]. The presence of a defect indicates a mismatch in what is developed and what was actually specified, thus affecting the quality of the end-product [20]. Intuitively, the count of defects is inversely proportional to the software quality of the deliverable released [20].

**Productivity** is used to gauge the quantity of output observed per unit input [21]. Programmer productivity is the amount of source lines of code produced by a programmer in unit time. PI is used to measure productivity relative to the rest of the industry. It is determined using the Single Lines of Code (SLOC) produced by each project, the effort in person-hours devoted towards the project and the project duration which is based on the Putnam formula for productivity calculation Size/Productivity = (Effort)^a X (Duration)^b [22]. Here ‘a’ and ‘b’ are exponents of effort and duration respectively their values are 1/3 and 4/3 respectively. Since PI represents productivity, the equation indicates that the PI for a project is directly proportional to its size and indirectly proportional to the duration of completion. A higher exponent on duration suggests a greater sensitivity of productivity.
towards the duration of project completion [22]. Any team that produces more lines of code in a shorter duration is considered to possess higher productivity.

5.1 Validation Data

We segregate our analysis based on the data obtained for quality and productivity metrics. Since, Claims 1 & 2 pertain to quality we provide the analysis in the quality section following which we evaluate productivity data for supporting Claim 3. The validation for Claim 4 requires the macro-level analysis of both quality and productivity data. Thus, it is attempted at the end.

5.1.1 Data for Quality Analysis

Quality data pertaining to an application has both the count of defects encountered in the process of its development and also the ones it is rolled out with. As discussed earlier an application is considered to have the highest of quality if it is released with no defects, which is only possible in an ideal scenario.

Applications that already exist in production need features to be added to them based on requirements from the business standpoint. Each application development team in the chosen ITSA caters to the development needs of an application. Defects encountered
during the SDLC are logged into the Quality Center [23] that acts like a bug tracker and a defect repository. Each defect logged into the repository contains details pertaining to 1) the release in which was found, 2) the application for which the release was scheduled, 3) the ITSA to which the application belongs, 4) the phase of SDLC in which it was detected, 5) the date on which the defect was detected, 6) classification of the defect, 7) severity of the defect, etc. Desired defect records can be pulled up based on the selection of filters associated with each of the above-mentioned fields.

A defect logged into the repository can be due to an issue caused by an infrastructure setup or a requirement misunderstanding, a flawed design or due to the application itself. Since, CI is linked to the development of applications the prime focus here is on defects arising due to application issues. In addition to this, it was essential to filter out defects that sometimes got logged more than once by testers and also the ones that were misunderstood by testers as defects and were termed as ‘not-a-defect’ kind of defects.

5.1.2 Data for Productivity Analysis

As discussed earlier, PI is calculated on a project basis where project specific information serves as the required parameters towards its determination. Among the limited areas within the organization that accurately report project specific information for PI calculations the Development Center (DC) is one of them. The DC constitutes of over 30
agile development teams working on different technologies while adhering to the industry best practices and collecting metrics for the same. Strong compliance to the industry best practices by the DC guarantees precision in the metrics collected and makes it a likely candidate for the productivity data analysis. Annual productivity metrics for JAVA/J2EE projects developed annually within the DC are plotted on control charts [24] which showcase how these agile development teams in the DC perform against the rest of the industry.

Referring to Figure 2, the nature of the relationship between an ITSA and DC is described as being matrixed. It implies that the effort that the DC devotes towards the development work catering to an ITSA is documented and archived in person-hours. In addition to effort, the defect data appearing while developing ITSA applications by the DC development teams are also logged in the repository that guarantees better traceability.

5.2 Analysis

5.2.1 Quality Metric Analysis
In order to prove the claims surrounding software quality and the fact that CI facilitates the detection of defects earlier in the phases of the Software Development Life Cycle a clear delineation between the phases considered as early and the ones considered as final is required. As shown in Figure 4 the development lifecycle in the organization has a list of phases right from Business Requirement/Analysis to Production. Since, CI is introduced in the Development/Unit Test phase of the life cycle and three stages constitute the Production segment we segregate phases from Development/Unit Test to Implementation as being the earlier phases, which we denote as pre-production and from Production Pilot to Production as the final phase production for ease of reference throughout the analysis.

**Figure 4** Phases in SDLC as observed in the enterprise
Since, every defect affects the quality of an application differently, defects are classified on the basis of severity and a weight is attached to each. The ones that need immediate attention are the ones that are tagged as Critical or High severity defects. The presence of such defects is detrimental to the functioning of an application and requires the fastest possible elimination. In addition to the above two kinds, there exists a medium severity defect type which implies a mismatch in the actual results with expected results that doesn’t have a significant impact in the functioning of related activities.

From the business point of view it is extremely beneficial to have almost zero occurrence of the above defects in an application that is rolled out. Even though a complete elimination of such defects cannot be guaranteed from a practical approach, decimation of its percentage in production can be made possible if these defects are detected early on which is facilitated by CI. Analysis of quality data corresponding to only the aforementioned defects extracted from the repository are used to verify \textbf{Claims 1} and 2.

We begin our analysis of the defect data that has been filtered on the basis of severity that now includes Critical, High and Medium defects alone arising solely out of application issues. Defect numbers are extracted for 6-month periods from 2011 to 2013. These 6-month periods signify the first and second halves of a calendar year. The rationale behind choosing these years lies in the fact that CI was introduced in the organization around the second half of 2011 and the repository has the capacity to hold data for a maximum period
of three years. Numbers for a total of six 6-month periods have been extracted out of the defect repository. These defects are selected so as to be contained within the phases starting from Development/Unit Tests to Production. This forms the basis of our analysis on which we calculate the percentage of defects showing up in the pre-production and the production phases. In a particular period irrespective of the number of defects arising in it if the percentage of those defects being found in production is higher than the ones being found in pre-production it indicates that more defects were found as the application was rolled out into production than in the phases prior to it. This defies the whole motive of the software development methodology that revolves around releasing the final software with the least possible defects. Since, CI claims to detect defects earlier in the phases of SDLC we can expect defect numbers to have higher percentage of defects in the pre-production phases rather than production. Also, in a given period the percentage of defects found in pre-production and production phases should sum to 100 percent. Ideally, we should observe a constant decline in the percentage of defects observed in production and an equal amount of increase in pre-production with the passage of time due to the symmetrical nature of their relationship. Likewise, the ratio of the percentage of defects in production to pre-production should show a steady decline.

We employ the above technique to capture the quality trend at the macro and micro levels in the enterprise. Defect data gathered for the ITSA corresponds to the macro-level quality data and for the micro-level analysis we focus on two applications named App-1 and App-
2 that exist within the ITSA. The applications are profiled on the basis of their nature of development activity. App-1 has had only maintenance kind of work throughout the period of the analysis and App-2 has had a combination of both maintenance and new development during the same period, the specifics of which are discussed in the analysis.

5.2.1.1 ITSA Macro-level Analysis

Defect numbers for the ITSA whose JAVA/J2EE application development teams have been practicing CI since its introduction into the organization are gathered on a half-yearly basis and their numbers are plotted with a stacked histogram in Figure 5A. The year 2011 is considered as the start year since CI was introduced into the organization around the second half of 2011. The dataset for the first half of 2011 is very small as the quality repository capacity limits the storage of historical defect data to just three years. Nonetheless, analyzing the data from the second half of 2011 we collect the total defect numbers, defect numbers found in production and prior to production in all the CI driven applications over each of the 6-month periods. The total defect numbers found in each of the 6-month time periods don’t show a steady decrease with time. Indicating that with a decrease in production defects there is an increase in the number of bugs being found in the earlier phases prior to production. With the defects in production reducing over the years we can say that deliverables are being released with less bugs than earlier. Thus, implying to an
increase in the overall quality of applications being developed by the ITSA. In addition to the defect numbers in the phases, in Figure 5B the defect percentages found in pre-production and in production are plotted to see if they suggest an improvement in quality. It turns out that starting from the second half of 2011 there is a significant increase in the occurrences of bugs found in the pre-production phases which is the same time around which applications starting being developed using CI within the ITSA. Thus, the graph in Figure 5B denoting the percentage of defects found prior to production shows an increase over the years. Hypothetically, if CI was being practiced to its full potential then the percentage of defects found prior to production should have reached 100 percent, implying zero production defects. This is not always possible in the real world as from interviewing Technical Leads and Iteration Managers of several application teams in the organization it has been found that the cause of defects in production is a result of the difference between actual production data and test data that is used to test applications in environments prior to production. The inability to accurately replicate the production environment during test phases introduces defects as applications are released into production. The interviews also revealed that if an application undergoes a complete rewrite then it leads to the occurrence of more production defects initially as the application is almost like a new one being deployed into production for the first time. This phenomena is captured in our quality analysis at the micro-level. Also, the control chart in Figure 5C plots the ratio of percentages of defects found in production to prior to production. This ratio is almost
always less than 1 as it is the sole motive of SDLC to produce deliverables that have the highest quality with the least amount of defects as they are rolled out to production. In addition to it if this ratio tends to show a steady decline with time then it is a clear indication of the fact that more defects are being found prior to production and less in production. However, this ratio should stay bound by the Upper and Lower Control Limits as an infringement would be an indicator of instability in the process. From the above graph it can be observed that we see a decline in this ratio and by the end of the second half of 2013 the ratio remains beneath the mean. The increase in pre-production defect percentage could be due to various other application development factors like the lesser complexity of applications, increased maintenance type development activity etc. We proceed to look at specific applications to further determine the cause of the above mentioned trend.

Figure 5 ITSA quality data and control charts
5.2.1.2 Application Micro-level Analysis

Relating Macro (ITSA) and Micro (App 1 and App 2) data.

**Figure 6 Top:** Control Chart differences  **Bottom:** Quality trend differences
Among the JAVA/J2EE applications that comprise of the ITSA we focus on two applications called App 1 and App 2 respectively. App 1 is a stable JAVA/J2EE application whose major development needs have been the addition of features to the already existing application in production. However, App 2 is a JAVA/J2EE web-application that underwent a complete rewrite in the second half of 2012. Both of the development teams started practicing CI in the second half of 2011 and thus we had longitudinal data available. For each of App 1 and App 2 we obtained pre-production and production defect data similar to that of the ITSA. Next, for each of App 1 and App 2, we developed the associated control chart, which is illustrated in Figure 6 Top. Along with that as mentioned earlier we also interviewed technical leads and iteration managers to obtain a profile of the application development activity. This is illustrated in Figure 6 Top as the application profile tables with M (for maintenance) and ND (for new development) entries. For each time interval, this table therefore characterizes the type of tasks that the respective project teams were involved in and corresponds to the control chart for that application.

The control charts indicate the ratio of defects found in production to pre-production. Ideally, this ratio should show a steady drop to support that CI improves quality. However, it does not exhibit this behavior. The application profile table is used to explain this.

The control charts together with the application profile table indicate that for App-1 there is an increase in the ratio going from the first half of 2011 to the second half. The reason here is the small size of the data set for the first half of 2011. Going forward from the
second half of 2011 we see as constant decline as the development activity remains to be of maintenance type. Although, the ratio for App-2 has a rise going into the second half of 2012 which is indicative of an increase in defects being detected in production. This degradation in quality corresponds to the application rewrite that takes place during the same period causing more defects to appear as the new application is pushed to production. Going forward, the rewrite process continues till the end of the first half of 2013. It is important to note that there is no further increase in the ratio. Thus the above analysis of the individual applications reveals a correspondence between development fluctuations and the defect ratio at the micro level. In Figure 6 Bottom the percentage of pre-production defects found in App 1, App 2, composite for App 1& App 2 and the ITSA are all plotted in a single graph for comparison.

Barring the first half of 2011, it is observed that the line graph for ITSA bounds the other line graphs from above. It is a smoother upward trend of improved production quality, than is evident in the individual applications as well as the composite of App-1 & App-2. This provides evidence that improvement in software quality is more pronounced at the macro-level since it has the composite trend of software quality being influenced by CI on multiple applications.

Relating Analysis to Claims: Summarizing, we have shown that there exists a trend in which defects are increasingly being detected in the phases prior to production, which supports Claim 1. The data also supports the corollary to Claim 1. That is with more defects
getting detected early on, fewer defects show up in production suggesting an increase in production software quality. Thus, supporting Claim 2.

To further strengthen Claims 1 and 2, we next provide support for Claim 4. If we look at the specific profiles of App 1 and App 2 for explanation of micro-level variations, we can see that there is a less pronounced yet explainable contributing trend at the micro level that supports the defect ratio improvement at the ITSA macro level. A stronger correlation between improvement in quality and maturity in the practice of CI exists at the macro-level compared to the application level. Hence, Claim 4 strengthens 1 and 2.

5.2.2 Productivity Metric Analysis

PI is the metric used to determine programmer productivity in a development team. As explained earlier any team that produces more Lines of Code (LOC) in a short duration is considered to possess higher productivity. The simplified formula from the Putnam equation that is used to compute PI values is \( PI = \frac{\text{Size of Project}}{(\text{Effort} \times \text{Duration})} \).

The research includes productivity metric data of JAVA/J2EE projects that the agile development teams in the DC of the organization worked on from the year 2011 through 2013. The DC of the organization is essentially an IT area that follows the industry best-practices and is CMMI level 3 certified. It is accountable for 40 percent of the development work in the enterprise in terms of effort defined as person-hours. The motive behind
choosing PI data from the DC is to observe if the productivity of the agile development teams within it increases as they continue to practice CI in their development ventures.

PI values have been plotted for all the JAVA/J2EE projects that the DC worked on each year. Since there are numerous projects every year the plot obtained is represented in a scatter graph using control charts as an evaluation strategy. Owing to a direct proportionality between the PI metric and Single Lines of Code produced by a project the lines in the control chart have a positive slope. Control charts are obtained from the data visualization feature available with the SLIM tool. For the purposes of maintaining a greater control on the process through which the metric data is obtained the Upper and Lower Control Limits are set to two standard deviations above and below the Mean line. In the control chart any point on the Mean line denotes the standard PI value for the rest of the industry for that particular SLOC value as projected by that point on the X-axis. A positive correlation can be inferred if there exists a higher Productivity Index value for the same region on the X-axis as CI practices mature with time in the DC.

We start with the analysis of PI data for the year 2011 and continue the same for the subsequent years where a comparison is drawn based on the spatial positioning of the PI values on the control chart.
2011 PI Data for JAVA/J2EE projects in DC

![PI vs Effective SLOC](image)

**Figure 7** PI data for DC projects for 2011

In **Figure 7**, the Y-axis denotes the PI values and the X-axis has the SLOC measured in Kilo Lines of Code. The mean PI of the total data is evaluated to 18.59 for an effective
SLOC of 32,190.44. The standard deviation (sigma) is at 4.06 PI and the minimum PI value in the data is 9.3 and the maximum 24.7. Close observation shows two points to be straying out of the Upper Control Limit. Rest of the projects stay well within the control limits validating the productivity of the system to be within stable limits.
2012 PI Data for JAVA/J2EE projects in DC

Figure 8 PI data for DC projects for 2012

The scatter plot in Figure 8 for 2012 has the PI values for 16 JAVA/J2EE projects that have been developed using Hudson in the DC. The mean PI is 20.67 with an effective
SLOC of 81,566.31 and the standard deviation is 3.49. Maximum PI observed here is 28 and the minimum 16.5. Like the 2011 scatter plot the PI values corresponding to two projects are observed to be above the Upper Control Limit. However, majority of the projects are bound within the upper control limit and the line one standard deviation above the mean. This indicates that the system now shows higher productivity within stable bounds.
Figure 9 PI data for DC projects for 2013
In addition to traditional JAVA/J2EE projects the year 2013 saw the introduction of Groovy [25] projects that integrated code following the standard CI practices. There are 20 projects in Figure 9 whose PI values are plotted out of which 9 projects stray outside of the upper control limit. Interviewing the Subject Matter Expert on metrics for the DC it was found out that the PI is highly sensitive to the duration in which a project is completed. A minor decrease in the completion time of a project can produce a huge increase in the calculation of productivity index. Here it is observed that PI values exceed the upper

Figure 10 Overall trend in PI data for all years

<table>
<thead>
<tr>
<th></th>
<th>Year 2011</th>
<th>Year 2012</th>
<th>Year 2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean PI</td>
<td>18.59</td>
<td>20.67</td>
<td>21.24</td>
</tr>
<tr>
<td>Median PI</td>
<td>18.99</td>
<td>20.98</td>
<td>21.79</td>
</tr>
<tr>
<td>Minimum PI</td>
<td>9.3</td>
<td>16.5</td>
<td>9.1</td>
</tr>
<tr>
<td>Maximum PI</td>
<td>24.7</td>
<td>28</td>
<td>32.9</td>
</tr>
</tbody>
</table>
control limit for projects between 1,000 and 10,000 SLOCs. The reason being that those projects had lesser completion times. Also focusing on the segment on the X-axis between 10,000 and 100,000 SLOCs we observe a drastic increase in the PI values from the previous years. This phenomena coupled with the fact that for a project with 2000 SLOCs the PI value obtained in 2011 was 9.3 and a 2,200 SLOC project in 2013 is 14.9 is a strong indicator of a steady increase in the productivity of the DC as a whole. The mean for the year 2013 had been set to 21.24 for an effective SLOC of 54, 843.45. The maximum and minimum PI values are found to be 32.9 and 20 respectively.

With the progression of time through which CI practices matured in the DC in Figure 10 a clear increase is seen in the mean PI in the annual Productivity plot from 18.59 in 2011 to 20.67 in 2012 and 21.24 in 2013. The maximum and median PI values for these years have also gone up drastically. However, the minimum PI values do not display a steady increase from 2012 to 2013. This can be reasoned to the PI value of project whose size was 350 SLOC and the direct proportionality of PI to size of project results in a dip in the minimum value of PI recorded for 2013. Focusing only on the PI values for the segment between 10 to 100 Kilo SLOCs on the X-axis a steep increase in productivity values is evident.
Relation to claims - From analyzing the annual data sets it was observed that projects that were sized between 10,000 to 100,000 SLOC showed a reasonable improvement in productivity as CI practices were followed within the DC, supporting Claim 3.

However, noting that PI has several contributing factors, to identify the causes for PI improvement that are pronounced at the macro level, we again need to drill down to the micro level. Due to the absence of similar sized projects for microanalysis across the three years for valid comparisons, we could not drill down and identify contributing factors from the micro to the macro level. This makes it difficult to leverage Claim 4 and provide additional insight for the existence of pronounced productivity improvements due to CI at the macro-level due to micro-level impact.

5.2.3 DC Quality Data Analysis

As mentioned earlier the DC of the Fortune 100 organization contributes to around 40 percent of the whole development in terms of person-hours that is carried out in the enterprise. Monthly metric reports from January to February and then April to June of 2014 from the ECTO’s (Enterprise Chief Technical Officer) office concerning quality and productivity confirm that the DC contributes to less than 10 percent of the production defects found in the total enterprise. In Figure 12 a trend is plotted to show the percentage of production defects that are found in the DC for the above mentioned months of 2014.
Figure 11 displays the aggregated monthly production defect numbers obtained for the whole enterprise from the ECTO’s office from January to June 2014 barring March. We have 16,523 defects from the whole enterprise out of which 1,144 production defects arise from the DC across those five months amounting to approximately 7 percent of the total defect numbers. Contributing towards 40 percent of the total development work in the enterprise and producing 7 percent of the total production defects we can predict a rough estimate of how many defects the DC would introduce if it were given the onus of
development from the rest of the enterprise too. If the rest 60 percent of the work was also through the DC then it would have added 11.5 percent to the existing number making it 18.5 percent. This hypothetical assumption provides an insight of how defects could reduce if the whole enterprise followed the best practices like the DC that majorly include CI.

![Production Defect Percentages in the DC](image)

**Figure 12** Production defect percentage by DC from Jan-June 2014

The above graph in **Figure 12** highlights the huge difference in the number of production defects contributed by the DC and the rest of the enterprise. Close observation shows that the percentage of contribution towards production defects does not even exceed 10 percent.
of the whole enterprise which is pretty impressive considering the fact that the DC accounts for about 40 percent of the total development work in the organization.

**Conclusion**- The quality data evaluation for the DC suggests an improvement in software quality at the macro-level which supports the **Claims 2 and 4**. In addition to that, it also supports **Claim 1** as the DC contributes to the least number of production defects in the whole enterprise which is indicative of the fact that the defects in the software development process is caught early on in the SDLC phases. Since, all the application development teams exhaustively practice CI in their development process the improvement in quality could be correlated to the maturity of their CI practices.
Chapter 6: Conclusion

The quality data analysis carried out on an ITSA level across all application development teams reveals that there exists a strong correlation between improvements in software quality with the passage of time indicating maturity of CI practices in the ITSA. In order to filter out noise introduced by the presence of data from non-CI application development teams, drilling down to two distinct applications contained within the ITSA it is found that they both exhibit the same trend with minor variations. The variance in their quality trends can be reasoned to the differences in the nature of development work carried out for the two applications. The application development team whose development activities involve releasing new features into the already existing application in production showed a steady increase in the software quality post introduction of CI and the one that had to go through a complete rewrite of the application showed variances in the software quality as their releases contained more defects in production phases. However, as both their CI practices reached maturity by the start of the year 2013 the defect numbers began decreasing for the deliverables being released into production implying improvement in software quality.

Macro level monthly quality data for the year 2014 obtained for the DC reveals that the development work carried out by the application development teams within it contributes to less than 10 percent of the total production defects arising in the whole enterprise. Ideally if the development practices were uniform across all the areas in the enterprise then doing
40 percent of the organization’s total development work should have amounted to contributing 40 percent of the production defects in the enterprise. DC is another area that has been practicing CI right since its introduction and going by their monthly defect numbers they have been delivering high quality software with the least number of defects. This acts as a strong correlation between maturity of CI practices and improvement in software quality. Hypothetically one can also infer that if the rest of the enterprise were adhering to practices similar to the DC then the whole enterprise would observe a drastic decimation in the production defect numbers. Mathematically, if the rest 60 percent of the work went through the DC then it would have contributed to less than 25 percent of the current total production defects in the enterprise. Not all the application development teams in the DC work on JAVA/J2EE but from a macro-level a reduction in production defect numbers across all teams satisfactorily implies a reduction in them too.

Observing the annual PI metric data for JAVA/J2EE projects shows a steady increase in their values from 2011 to 2013. Owing to the sensitivity of PI values to the SLOC of a project we tend to see a higher value for a project producing more lines of code. In order to make a fair comparison between PI data across all years we focus on the region on the X-axis between 10 and 100 Kilo SLOC. There is a significant increase in the PI values obtained in this region with the passage of time indicating an improvement in productivity with the maturity of CI practices. In addition to the cumulative PI values of projects we
tend to see an increase in the mean, median and maximum values obtained for each of the years in the line graph.

Basing on the analysis, the enterprise metric data is observed to have an improvement in them and this is successfully correlated with the maturity of CI practices. Thus, one can conclude on the basis of industry data that CI does improve software quality and productivity at a gradual pace.

There are also lessons learned. For example, testers only log defects into the repository after the code is moved through the various testing phases. Due to this practice, the integration and build issues that are detected through CI are never archived. This is because when the CI server notifies the developer of a build or integration fail, the developer rectifies the code causing it. If the developer records the cause for failures in the defect repository it would better support a causal quality analysis. Such causes for defects are invisible in current environment and cannot be considered in making claims.
References


50
Embedded Systems Conference Silicon Valley (San Jose, California), ESC 413, April 2007


[12] Hembrink, J., Stenberg, P-G. “Continuous Integration with Jenkins”, Coaching of Programming Teams (EDA 270), Faculty of Engineering, Lund University, LTH, March 5 2013


Appendix A: Quality Data

This section contains the defect data for App-1, App-2 and the quality trend of both the applications on which the analysis in Chapter 5 was based.

App-1 Quality Data

![APP-1 QUALITY DATA](image)

**Figure 13** App-1 raw quality from 2011-2013
The figure in **Figure 13** displays the total number of defects found in all the 6-month time frames from 2011 to 2013. The stacked histograms contain the total number of defects and the proportion of them appearing in the pre-production and production phases. The percentage of defects in production and pre-production phases are evaluated from the raw data above and visualized in the line graph below for App-1.

![Line graph](image)

**Figure 14** App-1 quality trend

The line graph in **Figure 14** visualizes the variation in the percentage of defects with time for App-1. Increase in the defect percentage in pre-production indicates that more defects are caught in the earlier phases and lesser probability of them appearing after the application is deployed in production.
App-2 Quality Data

**Figure 15** App-2 raw quality data 2011-2013

The above **Figure 15** contains the raw quality data displayed in the form of stacked histograms for all the 6-month time periods from 2011 to 2013. The line graph below calculates the percentage of defects from the raw data and provides the quality trend for App-2.
The line graph in Figure 16 shows an increase in the production defect numbers and a decrease in the pre-production ones in the first half of 2011 when CI was non-existent in the organization. Starting from the second half of 2011 the desired pattern begins to show and from 2012 that marks the beginning of the application rewrite shows the exact opposite of the desired pattern. However, as the initial ripples associated with the introduction of new changes into the system die out the pre-production defect numbers begin to pick up. Starting from the second half of 2013 the application rewrite is deemed complete and the development activity in the application begins to stabilize. It is during this period that CI seems to show a better correlation between its maturity and an improvement in the application quality as there is a drop in the defects found in production.

Figure 16 App-2 quality trend
App-1, App-2 Consolidated Quality Data

The data for both App-1 and App-2 were consolidated and a trend analysis was done to observe how quality improvements vary as more than one application starts benefiting from CI.
A better visualization of the production and pre-production defect percentages is given in Figure 18. There is an increase in the production defect percentages as CI is introduced around the second half of 2011, which could be attributed to the learning curve, associated with its introduction in both the teams. We see another spike in the production defect percentages around the second half of 2012 as this is around the time App-2 undergoes a complete application rewrite introducing more defects in production in their releases. From the start of 2013 we begin to see a decline in the production defects in both the applications and it continues to do so.

![App-1,2 Quality Trend](image)

**Figure 18** App-1, App-2 consolidated quality trend