Carnegie Mellon University
Software Engineering Institute

Composing Effective Software Security Assurance Workflows

William R. Nichols
James D. McHale
David Sweeney
William Snavely
Aaron Volkman

October 2018

TECHNICAL REPORT
CMU/SEI-2018-TR-004

Software Solutions Division

http://www.sei.cmu.edu

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors thank David Tuma and Tuma Solutions for donating the Team Process Data Warehouse and data import software for use our research involving TSP data. The Data Warehouse provides the technical infrastructure for the Software Engineering Measured Performance Repository (SEMPR). We also gratefully acknowledge the contributions of project data from the TSP user community and the developer of our data report queries.

Abstract

In an effort to determine how to make secure software development more cost effective, the SEI conducted a research study to empirically measure the effects that security tools—primarily automated static analysis tools—had on costs (measured by developer effort and schedule) and benefits (measured by defect and vulnerability reduction). The data used for this research came from 35 projects in three organizations that used both the Team Software Process and at least one automated static analysis (ASA) tool on source code or source code and binary. In every case quality levels improved when the tools were used, though modestly. In two organizations, use of the tools reduced total development effort. Effort increased in the third organization, but defect removal costs were reduced compared to the costs of fixes in system test. This study indicates that organizations should employ ASA tools to improve quality and reduce effort. There is some evidence, however, that using the tools could "crowd out" other defect removal activities, reducing the potential benefit. To avoid overreliance, the tools should be employed after other activities where practicable. When system test cycles require expensive equipment, ASA tools should precede test; otherwise, there are advantages to applying them after system test.

1 Introduction

This report describes the results of a research project called Composing Effective Secure Software Assurance Workflows (CESAW). The primary aim of the research was to investigate the impact of cybersecurity techniques and tools on software cost, schedule, and quality performance when they are applied throughout the software development lifecycle. The SEI collaborated with government and industry organizations that provided detailed performance data about their software projects for analysis and interpretation.

Section 1 provides the motivation and context for the CESAW research. Section 2 describes the research methodology. Section 3 presents the results of the research, and Section 4 explores these results and provides interpretation. Section 5 describes how the results can be used by software organizations and proposes additional research to further an understanding of this important topic.

1.1 An Economic Challenge for Cybersecurity

The economic challenges associated with fielding highly secure and cyber-resilient systems are well known [Baldwin 2011, Snyder 2015, DoD 2017]. Developing secure and cyber-resilient software requires multiple software security assurance (SSA) interventions throughout the development lifecycle. These interventions include manual methods (e.g., reviews and inspections) as well as automated methods (e.g., static analysis and dynamic analysis). There are now numerous SSA techniques and tools to choose from [Wheeler 2016]. Organizations must determine which specific SSA techniques and tools apply and decide when in the software development lifecycle to use them. However, despite a variety of models that are intended to address cybersecurity planning and implementation [Mead 2010, Howard 2007, Caralli 2010, Forrester 2006, Bartol 2008], the fundamental questions regarding the costs and benefits of SSA techniques are little understood. Larsen summarizes the problem by stating, “There is a general lack of relevant quantitative data about the true costs, schedule impact, and effectiveness (in various situations) of specific tools, specific techniques, and types of tools/techniques…This lack of quantitative data makes selecting tool/technique types, and selecting specific tools, much more difficult” [Wheeler 2016]. In the absence of guidance in the form of data or models, the selection and application of SSA techniques and tools is guesswork at best.

1.2 Modeling and Predicting Software Vulnerabilities

A number of researchers have reported results that are consistent with the thesis that a large portion of software security vulnerabilities result from common development errors. Heffley and Meunier reported that 64% of the vulnerabilities in the National Vulnerability Database (NVD) result from programming errors [Heffley 2004]. Martin summarized empirical findings from the Common Weakness Enumeration (CWE) that link vulnerabilities to common development issues [Martin 2014]. Shin and Williams reported empirical findings that 21.1% of the files in a web-based browser contained faults, and 13% of the faults were classified as vulnerability faults [Shin 2011]. Hence, the number of software-based vulnerabilities in an application can be correlated with the number of software defects in that application. That is, we make the assumption that if software defects are removed based on the application of an SSA intervention, then we can assume that cybersecurity-based defects are also being removed from the software application.

As part of this study, modeling techniques were used to evaluate the impact of defect-removal activities on the reduction of software defects. Nichols developed a cost of quality model that is used to inform software development planning decisions [Nichols 2012]. Specifically, the model predicts the amount of effort that must be applied on defect removal activities to achieve a specific level of quality. The model is parameterized using software size estimates and historical performance data. The model calculates expectation values for effort and defect densities. During the planning phase, a software team can examine quality, cost, and schedule goals by adjusting their plan and evaluating the consequences using the model.

Building on the existing cost of quality model, the SEI team engaged and collaborated with organizations that have integrated SSA tools into their software development lifecycle. The objective was to adapt the model by including actual parametric data resulting from the inclusion of SSA techniques and tools in the projects' development lifecycles and then assess the impact of the interventions on cost, schedule, and quality.

1.3 Software Development Approaches

For this study, the SEI used data from 35 projects from three organizations that develop software in avionics, design automation, and business intelligence. All of the projects developed software using the Team Software Process (TSP) approach. The TSP approach to software development was developed at the Software Engineering Institute (SEI), a federally funded research and development center at Carnegie Mellon University [Humphrey 1999].

1.4 Static Analysis Tools

As noted in Table 1, the participating organizations used different static analysis tools, and they also chose to use them during different phases of the lifecycle.

Table 1: Static Analysis Tools and the Process Phases Where They Were Used

Organization	# Projects	Domain	Static Analysis Tool	Phase Where Tool Used
A	5	Avionics	Tool A Static Code	Personal Review
B	16	Business Intelligence	Tool B_1 Static Code Tool B_2 Static Code and Binary	Compile, Code Inspection, Personal Review
C	14	Design Automation	Tool_C Static Code and Binary	Acceptance Test

Following is a brief description of the static analysis tools that were used on the projects in this study.

Static Analysis Tool A

Static Analysis Tool A is a static branded code analysis tool used to identify security, safety, and reliability issues in C, C++, Java, and C# code.

Tool_B_1

Tool_B_1 enforces a common set of style rules for C# code using a single, consistent set of rules, with minimal rule configuration allowed. Developers can implement their own rules if they so choose.

Tool_B_2

Tool_B_2 is an application that analyzes managed code assemblies (code that targets the .NET Framework common language runtime) and reports information about the assemblies, such as possible design, localization, performance, and security improvements.

Tool_C

Tool_C is a branded software development product, consisting primarily of static code analysis and static binary analysis. It enables engineers to find defects and security vulnerabilities in source code written in C, C++, Java, C#, and JavaScript.

1.5 Development Lifecycle

TSP projects use the software lifecycle activities that are listed in Table 2. These activities are considered the primary logical phases through which a software component or a change to a feature must pass. Please note that this does not imply a sequential approach without iteration. Rather, it characterizes the activities that should be performed on the product for each iterative cycle. Any phase can include defect injection or removal. However, each phase is characterized as being one where defects are primarily injected or removed. The *Creation* phase type indicates a phase where something is developed and defects are typically injected. The *Appraisal* phase type is one in which developers or technical peers examine the product and discover and remove defects. The phase type *Failure* is a phase where the product is tested and defect symptoms are identified. Developers must then isolate the defect causing the symptom and fix it. Defect removal through appraisal or failure typically incurs different costs. Table 2 indicates the phase type for typical TSP phases.

Table 2: Typical Phases of a TSP Project

TSP Phases	Description	Phase Type
DLD	Detailed-level design	Creation
DLDR	Personal review of the detailed design	Appraisal
TD	Unit test case development	Creation
DLDI	Peer inspection of the detailed design	Appraisal
Code	Writing the source code	Creation
CR	Personal review of the source code	Appraisal
CI	Peer inspection of the source code	Appraisal
UT	Developer unit test execution	Failure
IT	Integration test	Failure
ST	System test	Failure
UAT	User acceptance test	Failure
PL	Product life	Failure

The TSP approach has many characteristics in common with Agile projects. While there are a number of distinctions that set TSP apart from Agile, the most significant difference is the focus on quality and the inclusion of a measurement framework that makes it possible for software engineers to improve their performance. The TSP approach is defined in a series of process scripts that describe all aspects of project planning and product development. Within the scripts, operational definitions specify how measures are defined, estimated, collected, reported, and analyzed. The data that is used in this study is a compilation of data that was recorded in real time by software engineers as they conducted their work.

Table 3 lists the measures that are collected by teams using the TSP approach [Humphrey 2010]. In addition to the base measures, additional measures derived from them can provide insight into team performance in terms of cost, schedule, and quality.

Measures are collected in real time throughout the project and analyzed on a daily basis to guide and improve performance. This is accomplished using an automated tool called the Process Dashboard [SPDI 2017]. Each member of a TSP team enters their personal performance data into the Process Dashboard. The entries are compiled and combined automatically into a Team Dashboard that provides a daily accounting of overall team performance throughout the development lifecycle.

Table 3: Product Size Measures Collected Using the TSP Approach

Size Data	Quality Data	Schedule Data
Added product size	Defect ID	Task time and phase; product/element involved
Added and modified product size	Defect type	Task commitment date and task completion date
Base product size	Phase where defect was discovered	
Modified product size	Phase where defect was removed	
New reuse product size	Defect fix time	
Modified product size	Brief description of defect	
New reuse product size		
Reusable product size		
Reused product size		
Total product size		

2 Research Approach

Performance was evaluated using the cost of quality model that was briefly discussed in Section 1.2 (see "Plan for Success, Model the Cost of Quality" [Nichols 2012] for a detailed description of the model). For each of the three organizations, performance was evaluated with the use of the static analysis tool included and then compared to the hypothetical case in which the static analysis tool was not used.

2.1 Approach Background

According to Runeson, research serves four distinct purposes [Runeson 2012]:

1. Exploratory—finding out what is happening, seeking new insights, and generating ideas and hypotheses for new research.
2. Descriptive—portraying a situation or phenomenon.
3. Explanatory—seeking an explanation of a situation or a problem, mostly, but not necessarily, in the form of a causal relationship.
4. Improving—trying to improve a certain aspect of the studied phenomenon.

Since our research includes elements of improvement for quality, security, and cost, we adopt methods from software process improvement. Other important aspects include describing the phenomena for use in benchmarking and modeling and exploring how the tools are used in practice and describing the use quantitatively. Explanation is not a priority for this work. Our focus is on real-world application of the tools rather than use under ideal conditions. Research on real-world issues includes a trade-off between level of researcher control and realism. This is essentially a tradeoff between internal validity and external validity; this tradeoff has been discussed in medical effectiveness studies (as opposed to efficacy studies) [Singal 2014, Fritz 2003, Westfall 2007]. In other words, we do not seek to measure the efficacy of these tools in finding vulnerabilities; instead we want to evaluate how the use of these tools under real world conditions affects development. Therefore, our work is designed to fill a research gap in this aspect of external validity. In designing our approach to the research questions, we draw on experience from software process improvement, case studies, and medical literature on effectiveness studies.

2.1.1 Software Process Improvement

Software process improvement (SPI) is a systematic approach to increase the efficiency and effectiveness of software development. Because our research objective is at least partially aligned with the goals of SPI, we examined approaches for evaluating proposed process improvements. A systematic review of SPI literature evaluated 148 papers and summarized the approaches used [Unterkalmsteiner 2012]. The following approaches were found (ordered by frequency from most used to least):

- pre-post comparison
- statistical analysis
- pre-post comparison and survey

- cost/benefit analysis
- pre-post comparison and cost analysis
- statistical process control
- statistical analysis and survey
- software productivity analysis
- cost/benefit analysis and survey

We will briefly discuss some of these approaches, including their requirements and weaknesses.

2.1.1.1 Pre-Post Comparison

Pre-post comparison compares the value of pre-identified success indicators before and after the SPI initiatives took place. For a pre-post comparison of success indicators, it is necessary to set up a baseline from which the improvements can be measured. The major difficulty here is to identify reasonable baseline values against which the improvements can be measured [Rozum 1993].

One strategy could be to use the values from a representative successful project as the benchmark. An example that illustrates how to construct a baseline for organizational performance is provided by Daniel Paulish and colleagues [Paulish 1993, 1994]. A challenge to this approach, however, is the stability of the process benchmarks and wide variation [Gibson 2006].

2.1.1.2 Statistical Analysis

The statistical techniques presented in “Quantitative Evaluation of Software Process Improvement” [Henry 1995] can be used to create baselines of quality and productivity measurements. The statistical analysis includes descriptive statistics summarizing the numeric data (e.g., tables of the mean, median, standard deviation, interquartile range, and so forth) or graphically (e.g., with histograms, box plots, scatter plots, Pareto charts, or run charts). Inferential statistics can generalize representative samples to a larger population through hypothesis testing, numerical estimates, correlation, and regression or other modeling.

2.1.1.3 Cost/Benefit Analysis

Evaluating an improvement initiative with a cost/benefit measure is important since the budget for the program must be justified to avoid discontinuation or motivate broader rollout [Kitchenham 1996, van Solingen 2004]. Furthermore, businesses need to identify efficient investment opportunities and means to increase margins [van Solingen 2004]. When assessing cost, organizations should also consider resources beyond pure effort (which can be measured with relative ease); for example: office space, travel, computer infrastructure [van Solingen 2004], training, coaching, additional metrics, additional management activities, and process maintenance. Nonetheless, activity-based costing helps to relate certain activities with the actual effort spent [Ebert 1998].

Since actual cost and effort data can be collected in projects, they should be used. A useful technique to support estimation is the “what-if-not” analysis [Ebert 1998]. Project managers could be asked to estimate how much effort was saved due to the implemented improvement in follow-up projects. In our research, rather than use a subjective estimate, we used actual collected data to calibrate models for process variants.

2.1.1.4 Statistical Process Control

Statistical process control (SPC), often associated with time series analysis, can provide information about when an improvement should be carried out and help determine the efficacy of the process changes [Caivano 2005, Hare 1995]. SPC is often used to identify trends or outliers [Paulk 2009]. SPC can also be used to identify and evaluate stability using shape metrics, which are analyzed by visual inspection of data that is summarized by descriptive statistics (e.g., histograms and trend diagrams) [Schneidewind 1999].

2.1.1.5 Historic SPI Study Weaknesses

A number of common deficiencies were found in the literature [Unterkalmsteiner 2012]:

- Incomplete context descriptions were used that did not contain a complete description of the process change or the environment. The importance of context in software engineering is emphasized by other authors [Petersen 2009; Dybå 2012, 2013].
- Confounding factors were rarely discussed; frequently, multiple changes were introduced at once, presenting challenges for evaluation validity.
- Imprecise measurement definitions resulted in many problems, including broad ranges for interpretation [Kaner 2004].
- Scope was lacking beyond pilot projects. The effects on business, wider deployment, and fielded products were rarely discussed.

2.1.1.6 Hybrid Methods Using Participant Surveys

Our data collection was previously approved for use in research, but any surveys we conducted would require further approval from our institutional review board (IRB) for research involving human subjects. In light of time and project constraints, we decided to avoid the risk by foregoing the use of participant surveys.

2.1.2 Case Studies, Quasi-experiments, and Action Research

Research approaches in the literature include case studies, quasi-experiments, action research, project monitoring, and field study. Although the characteristics of these approaches have overlapping and sometimes shifting definitions, the guidelines for conducting case studies [Runeson 2008], can be applied to all of these approaches.

We had intended to apply the action research by helping the subjects implement both tools and measurement. We were unsuccessful in obtaining sufficient cooperation to introduce changes, however, so we reverted to observation of their currently implemented processes.

2.1.3 Effectiveness vs. Efficacy

Despite laboratory demonstrations of effectiveness, there remain a number of threats to the real-world effectiveness of tools and techniques. Scientifically valid tests can impose selection criteria on the subjects of the study, exert considerable control over the execution, and take place using highly skilled specialists who may not be generally available. Following the example of the medical industry, we distinguished between efficacy (explanatory) and effectiveness (pragmatic) trials [Fritz 2003]. The goal of efficacy trials is to determine how a technique, tool, or treatment works under ideal circumstances, which requires minimization of confounding factors. In practice, protocol deviations, other techniques, compliance, adverse events, and so forth can affect efficacy. Effectiveness studies evaluate the usefulness under real world conditions (i.e., in less than ideal situations or when tools are inexpertly applied).

Criteria that distinguish efficacy from effectiveness studies include the following [Gartlehner 2006]:

1. The setting is more or less representative of the state of the practice. The setting should not include extraordinary resources, equipment, training, or specialized skills.
2. The subjects should be representative of the general population with few restrictions. Selection and randomization enhance internal validity at the expense of generalizability.
3. The measure of success is the final outcome unless empirical evidence verifies that the effects on intermediate points fully captures the net effect.
4. Durations should mimic real application under conventional settings. Those implementing the techniques should exercise their judgment rather than be restricted by research protocols. Moreover, those judgments should reflect a primary concern for achieving project outcomes rather than satisfying the needs of explanatory research.
5. There should be an assessment of adverse events to balance benefits and risks. In effectiveness studies, for example, discontinuation rates and compliance are a feature, not a bug.
6. Adequate sample size is needed to assess a minimally important difference.

2.2 Study Design

In preparing this study we faced several constraints:

1. Without prior fundamental research approval, we were limited in what we could share with potential collaborators.
2. Without prior approval from the IRB for research involving human subjects, we were reluctant to include participant surveys as part of our design. This increased our reliance upon the software development process data.
3. Funding was limited to a single fiscal year, with the potential for modest extensions of unused funds.
4. We needed data that could answer our research question using our proposed technique, thus limiting potential collaborators.
5. Collaborators had to be willing to instrument their project to collect the required data.

2.2.1 Participant Selection

To address the fundamental research concerns we adopted two strategies. The first involved working with DoD development teams directly. The second was to identify suitable projects with data in the SEMPR repository.

The DoD collaborators were chosen by convenience: Because we had previously worked with these groups, we had existing relationships to leverage. For the other group we searched the SEMPR task logs, time logs, and defect logs for keywords associated with static analysis tools. We also sent email to TSP partners asking for projects that used these tools.

In SEMPR, we selected projects that met the following criteria:

- Multiple projects from the same organization were available.
- Tool use could be reliably determined from the log data.
- Project data without the tools existed.
- Data was complete (i.e., it included size, time logs, and defect logs).

One DoD project provided us with the data requested. The others offered test data, but declined to help with instrumenting the tool use. Because of the data gaps, we were only able to analyze the projects that sent the TSP data.

2.2.2 Data Collection

The primary data collection consisted of developer entries into the Process Dashboard tool. The DoD project data was sent to us in September 2017 following the end of the most recent development cycle. The data included prior cycles.

The other project data was submitted to the SEI through the Partner Network, meeting the terms of TSP licensing. The data included the Process Dashboard data files, project launch records, project meeting notes, and post-mortem reports.

2.2.3 Data Storage

Project data was stored on a secure SEI file system. The Dashboard files were imported into the SEMPR Repository [Shirai 2014] using the Data Warehouse [SPDI 2014]. After extraction, data was collected into fact sheets for summary [Shirai 2015].

2.2.4 Data Analysis

The data analysis included the following steps:

1. data extraction
2. data cleaning
3. statistical analysis
4. establishing baselines
5. model building

2.2.4.1 Data Extraction

Our first step was to identify projects of interest. For the DoD projects we traced the Process Dashboards to project IDs in the SEMPR repository. For the others, we first identified projects of interest based on the key words and found two organizations with sufficient data. We then identified specific project IDs associated with the static analysis tools. We verified our selections by

- reviewing the Project Dashboard data directly
- examining other project artifacts to confirm our assumptions
- holding discussions in person and by email with project participants to confirm tool usage and context

We then proceeded to data extraction. The Data Warehouse is built on a Microsoft SQL server. We developed scripts to query the database to extract the data needed to compute the parameters.

2.2.4.2 Data Cleaning

Although TSP data is of generally high quality, we needed to perform several steps to clean the data. First, we adjusted the development phases to a common baseline. Individual projects sometimes adapt the frameworks or use phases out of true order. We verified phase order with time log entries and discussions with project members. For one organization we combined phases into a single requirements phase and adjusted for some process name changes. For another organization we noted that “acceptance test” preceded system test and was used as part of the build.

Second, our measure of size includes “added” and “modified.” In practice, this number should be derived from base, total, added, and deleted. Some projects, however, entered this value directly.

The output fact sheets are provided as a research artifact available online in the SEI Library to those who would like to repeat this research or apply the data to other research questions. We also provide fact sheets that include the work breakdown structure coding, providing data at the component rather than the aggregated project level.

2.2.4.3 Statistical Analysis

To prepare for modeling, we extracted the direct data (size, phased effort, defects injected, and defects removed) and aggregated it to the project level. From there we derived phase injection rates, phase removal rates, phase defect fix times, phase performance rates, phase defect removal yields, phase defect densities, phase defect removal densities, and phase development rates with the rework time (defect find and fix effort removed). We collected parametric and non-parametric parameters and compared them to identify baseline projects.

2.2.4.4 Modeling

Using the project parameters for the organization averages, we applied our TSP quality model (similar to COQUALMO) [Nichols 2012, Madachy 2008]. The most significant adjustment here was to restrict the range of allowable ranges in system test.

Because some parameter uncertainties were large, we compared them with alternative scenarios in which we set the static removal phase effort to zero and the yield to zero. We then compared the resulting efforts and escape defect densities. Note that removing the tool allows more defects to escape into later phases. The model assumes a fixed yield in these phases so that overall defect escapes increase, but some of those defects are captured and removed in the later test phases. Depending on fix times and capture levels, this could potentially increase overall time to delivery. The result of this is an estimate of the net cost of applying the tool and the benefit as measured by defect density after system test.

3 Data Collection and Processing

Source data for this study was provided by 35 projects from three organizations that develop software in avionics, design automation, and business intelligence (see Table 4).

Table 4: Participating Organizations and Projects

Organization	# Projects	Domain	Total LoC	Total Defects
A	5	Avionics	641305	22160
B	16	Business Intelligence	118933	10676
C	14	Design Automation	178532	40542

Detailed data for each project are provided in the CESAW_Project_data.csv fact sheet (available online in the SEI Library), which includes information for each project (total added and modified lines of code) and each development cycle process (effort, defects found, defects removed, defect find and fix time). Project summaries are included in Table 5.

Table 5: Project Summary Data

Org	Project Key	Team Size	Start Date	End Date	A&M [LoC]	Effort [Hours]	Duration [Days]
A	615	48	8-Sep	14-Oct	796887	35091.5	2215
A	613	35	13-May	16-Mar	117279	7130.8	490
A	614	30	14-Jun	15-Jul	246118	7746.8	391
A	612	36	15-Jul	16-Nov	89127	10927.9	490
A	617	41	16-Apr	17-Jul	84316	10851.6	457
B	180	16	11-Jun	12-Feb	20318	2626.3	246
B	49	11	12-Jan	12-Dec	22411	1929.0	327
B	181	8	12-Jan	13-Jul	37123	3950.7	552
B	47	13	12-Jul	12-Aug	484	537.4	47
B	48	13	12-Jul	12-Aug	1865	707.9	47
B	606	12	12-Jul	12-Oct	4020	1278.4	88
B	50	15	12-Aug	14-Dec	6089	2248.8	844
B	56	4	12-Sep	13-Feb	0	749.4	148
B	182	7	12-Sep	12-Nov	4494	924.5	53
B	183	9	12-Nov	13-Jul	5148	1234.0	264
B	184	7	12-Nov	13-Aug	38302	3165.5	272
B	70	10	13-Feb	13-May	442	788.5	92
B	71	6	13-Feb	13-May	0	516.8	98
B	72	5	13-Feb	13-May	0	621.5	85
B	83	11	13-Apr	13-Aug	0	1334.2	112
B	84	4	13-May	13-Aug	0	556.6	100
C	23	3	11-Sep	11-Oct	23	21.2	21
C	456	22	12-Feb	13-Jul	2554	3207.8	512

Org	Project Key	Team Size	Start Date	End Date	A&M [LoC]	Effort [Hours]	Duration [Days]
C	455	19	12-Dec	13-Dec	737	572.2	374
C	458	20	13-Jul	13-Nov	0	2428.0	138
C	415	5	13-Sep	14-Mar	4042	815.8	178
C	459	20	13-Nov	14-Apr	83	1296.8	129
C	416	8	14-Jan	14-Apr	9678	1282.8	91
C	419	7	14-Jul	14-Oct	13333	1532.2	114
C	420	8	14-Nov	15-Jan	9741	1282.5	73
C	171	19	12-Jul	12-Dec	1817	2294.6	149
C	79	9	13-Jan	13-May	8998	1941.9	140
C	449	11	14-Jan	14-Nov	6500	2253.7	316
C	418	8	14-Apr	14-Jul	7806	1141.9	72
C	460	25	14-Apr	14-Oct	66499	3294.8	180
C	461	25	14-Sep	15-Jan	46694	3392.4	137

3.1 Process Activity Mapping

A traditional TSP development approach was briefly presented in Table 2. However, projects typically customize the process to fit their particular circumstances (e.g., whether systems engineering phases are included, inclusion of requirements definition phases or documentation phases, etc.). Although there is process customization, a key principle is followed: a quality assurance phase always follows a process phase in which product is created. That is, each product creation process phase is followed by an appraisal process.

The three organizations that are part of this study chose to customize the TSP development approach in various ways. Projects within each organization used their organization's customized TSP lifecycle definition. For the purposes of this study, it was necessary to use a common development lifecycle framework so comparisons could be made among the projects from different organizations. Therefore, the customized framework used by each project in this study was mapped to a standard framework that is presented in Table 6. The *Process Type* column lists the lifecycle type for the phase name in the first column. The *Phase Type* column distinguishes the purpose of the phase by one of four attributes: overhead, construction, appraisal, or failure.

Table 6: Standard Process Phases Mapped to Process Type and Phase Type

Standard Lifecycle Activity Name	Process Type	Activity Type
Launch and Strategy	Strategy	Overhead
Planning	Planning	Overhead
System Engineering Requirements Identification	System Requirements	Construction
System Engineering Requirements Identification Inspection	System Requirements	Appraisal
System Engineering Requirements Analysis	System Requirements	Appraisal
Launch and Strategy	Strategy	Overhead
System Engineering Requirements Review	System Requirements	Appraisal
System Engineering Requirements Inspection	System Requirements	Appraisal

Standard Lifecycle Activity Name	Process Type	Activity Type
System Engineering Test Plan	System Requirements	Construction
System Design	System Requirements	Construction
System Design Review	System Requirements	Appraisal
System Design Inspection	System Requirements	Appraisal
Software Requirements analysis	Software Requirements	Construction
Software System Test Plan	Software Requirements	Construction
Software Requirements Review	Software Requirements	Appraisal
Software Requirements Inspection	Software Requirements	Appraisal
High-Level Design	High Level Design	Construction
Integration Test Plan	High Level Design	Construction
HLD Review	High Level Design	Appraisal
HLD Inspection	High Level Design	Appraisal
Detailed Design	Detailed Level Design	Construction
Unit Test Development	Detailed Level Design	Construction
Detailed Design Review	Detailed Level Design	Appraisal
Detailed Design Inspection	Detailed Level Design	Appraisal
Code	Coding	Construction
Code Review	Coding	Appraisal
Compile	Coding	Failure
Code Inspection	Coding	Appraisal
Unit Test	Module Test	Failure
Independent Test Plan		Construction
Build and Integration	Integration Test	Failure
Functional Test		
Software System Test	Software System Test	Failure
Documentation		Construction
Acceptance Test		Failure
Postmortem		
Transition and Deployment		Construction
After Development		
Product Life		Failure
Other Test Plan		
Other Test Plan Review and Inspect		
Other Test Development		
Other Test Case Review and Inspect		
Other Testing		

Table 7 illustrates how the lifecycle phases of each organization are mapped to the standard lifecycle phases.

Table 7: Project Phases Mapped to Standard Lifecycle Phases

Standard Lifecycle Phase Name	Organization A	Organization B	Organization C
Management and Miscellaneous	Management and Mis-cellaneous	Management and Mis-cellaneous	Management and Mis-cellaneous
Launch and Strategy	Launch and Strategy	Launch	Launch and Strategy
Planning	Planning	Planning	Planning
System Engineering Requirements Identification		Problem Identification	
System Engineering Requirements Identification - Inspection		Problem Identification Inspection	
System Engineering Requirements Analysis		In Work	
System Engineering Requirements Review		Work Inspection - Author	
System Engineering Requirements Inspection		Work Inspection - Others	
System Engineering Test Plan		Integration Test	
System Design			
System Design Review			
System Design Inspection			
Software Requirements analysis	Requirements	Requirements	Requirements
Software System Test Plan	System Test Plan	System Test Plan	
Software Requirements Review	Requirements Review	Requirements Review	
Software Requirements Inspection	Requirements Inspec-tion	Requirements Inspec-tion	Requirements Inspec-tion
High-Level Design	High-Level Design	High-Level Design	High-Level Design
Integration Test Plan	Integration Test Plan		
HLD Review	HLD Review		HLD Review
HLD Inspection	HLD Inspection	HLD Inspection	HLD Inspection
Detailed Design	Detailed Design	Detailed Design	Detailed Design
Unit Test Development	Test Development	Detailed Design Re-view	
Detailed Design Review	Detailed Design Re-view	Test Development	Detailed Design Re-view
Detailed Design Inspection	Detailed Design In-spection	Detailed Design Inspection	Detailed Design In-spection
			Unit Test Development
Code	Code	Code	Code
Code Review	Code Review	Code Review	Code Review
Compile	Compile	Compile	Compile

Standard Lifecycle Phase Name	Organization A	Organization B	Organization C
Code Inspection	Code Inspection	Code Inspection	Code Inspection
Unit Test	Unit Test	Unit Test	Unit Test
Independent Test Plan			
Build and Integration	Build and Integration Test	Build and Integration Test	Build and Integration Test
Functional Test			
Software System Test	System Test	System Test	System Test
Documentation	Documentation	Documentation	
Acceptance Test	Acceptance Test		Acceptance Test
Postmortem	Postmortem		Postmortem
Transition and Deployment			Product Life
After Development			
Product Life	Product Life	Product Life	Documentation
Other Test Plan			Documentation Review
Other Test Plan Review and In-spect			Q-Test Planning
Other Test Development			Documentation Inspec-tion
Other Test Case Review and In-spect			Q-Manual Test Case Design
Other Testing			Q-Manual Test Case Development
			Q-Manual Test Case Review
			Q-Manual Test Case Inspection
			Q-Auto Test Case De-sign
			Q-Auto Test Case De-velopment
			Q-Auto Test Case Re-view
			Q-Auto Test Case In-spection
			Do Not Use - Integra-tion Test Plan
			Do Not Use - System Test Plan

3.2 Data Collection

Organizations recorded their own data using the Process Dashboard [Shirai 2014] while using the Team Software Process to plan and track their software projects. The data collection consists of logging all direct effort on project tasks, recording defects detected and repaired, and recording product size as measured in lines of code.

Project data from each organization included the variables described in Table 8 through Table 11. The project data from each organization were combined and averages were used for the performance comparisons.

3.3 Data Definitions

This section describes the data variables that were evaluated for the performance analysis.

Table 8: Variables Identifying the Source Data

Variable	Description
project_id	Unique numerical identifier of the project
organization_key	Unique numerical identifier of the project's organization
team_key	Unique numerical identifier of the project team
wbs_element_key	Numerical assignment that identifies the work breakdown structure element to which each data record applies

Table 9: Variables Used For Recording Product Size Information

Measure	Description
size_metric_short_name	Abbreviation for the size measure that is being used for product size. Examples include lines of code (LOC), pages, and use cases.
size_added_and_modified	Number of new units of size that are added to a new or existing product
size_added	Number of new units of size that have been added to the product
base_size	Number of units of size already existing in a product before it is modified by the developer to arrive at the new product
size_deleted	Number of units of size that are deleted from an existing product during modification by the developer to arrive at the new product
size_modified	Number of units of size of existing product that are modified to arrive at the new product
size_reused	Number of units of size that are copied from a library or repository "as-is" and included in the product under development

Table 10: Variables Associated with Software Product Quality

Measure	Description
Cumulative Defect Injections	Sum of defects injected in the product during the current and all previous product development phases
Cumulative Removals	Sum of defects removed in a product during the current and all previous product development phases
Defect Density	Number of defects detected in the product during a phase divided by the size of the product
defect injection rate	Number of defects injected into the product per hour
Defect_fix_time	Total number of task time minutes required to fix all discovered defects within a process phase
Defects Injected	Total number of defects injected into a product during a process phase
Defects Removed	Total number of defects removed during a process phase
Development Effort	Total number of task time minutes in a process phase
phase effort per_defect	Number of task time hours associated with finding and fixing errors for each defect
Phase_Escapes	Within a given process phase, the cumulative removals minus the cumulative defect injections
Phase_no.Defect_cost	Total number of task time minutes within a process phase applied to product development, minus the task time minutes associated with finding and fixing defects within that phase (Phase_no.Defect_cost =Total.Phase.Effort - sum(defect.Find&Fix_Effort)
Phase_Rate_no.defect_effort/LOC	Total number of task time hours per LOC associated with product development minus the task time hours per LOC associated with finding/fixing product defects
Size	Size of the product measured in lines of code
Yield	Percentage of product defects that are removed during a process phase

Table 11: Variables Associated with Development Effort

Measure	Description
0000_BeforeDev_act_time_min	Task time applied to project before software development begins
1000_Misc_act_time_min	Miscellaneous task time applied to project activities. Some TSP teams used this category to track task time that is not directly applied to development activities.
1100_Strat_act_time_min	Task time applied to developing the strategy for the project
1150_Planning_act_time_min	Task time applied to planning the project
1200_SE_REQ_act_time_min	Task time applied to systems engineering requirements definition/analysis
1220_SE_REQR_act_time_min	Task time applied to reviewing and fixing defects in the requirements
4030_STest_act_time_min	Task time applied to developing the system test plan
4040_Doc_act_time_min	Task time applied to developing documentation (e.g., installation manuals, user guides, etc.)
4050_ATest_act_time_min	Task time applied to developing the acceptance test plan for the product
5000_PM_act_time_min	Task time applied to post mortem (lessons learned) activities throughout the project
6100_PLife_act_time_min	Task time applied during the product life phase (following product release)
6200_AfterDev_act_time_min	Task time applied after product development but before product release
0000_BeforeDev_fix_time	Task time applied to finding and fixing defects in an existing product before it is enhanced
1000_Misc_fix_time	Placeholder for fix time data that did not map to other process phases
1100_Strat_fix_time	Task time applied to making corrections or changes to the current strategy for the project
1150_Planning_fix_time	Task time applied to making changes to the software development plan
1200_SE_REQ_fix_time	Task time applied to making corrections or additions to the systems engineering requirements during the requirements process phase
1220_SE_REQR_fix_time	Task time applied to personal review of the requirements, and finding and fixing any errors during the systems engineering requirement process phase
1240_SE_REQI_fix_time	Task time applied to multi-person review of the requirements, and finding and fixing any errors during the systems engineering requirements inspection phase
1250_SE_REQ_Val_fix_time	Task time applied to validation of the requirements and finding and fixing any errors during the systems engineering requirements validation phase
3000_Req_fix_time	Task time applied to finding and fixing any software defects during the requirements phase of the project
3020_ReqR_fix_time	Task time applied to finding and fixing software defects during the requirements review phase of the project
3040_ReqI_fix_time	Task time applied to finding and fixing software defects during the requirements inspection phase of the project
3100_HLD_fix_time	Task time applied to finding high-level design defects during the high-level design development phase of the project
3110_ITP_fix_time	Task time applied to finding and fixing defects during the integration test planning process phase
3120_HLDR_fix_time	Task time applied to finding and fixing design defects during the high-level design review process phase

Measure	Description
3140_HLDI_fix_time	Task time applied to finding and fixing detailed design defects during the high-level design inspection phase
3200_DLD_fix_time	Task time applied to finding and fixing detailed design defects during the design phase
3210_TD_fix_time	Task time applied to finding and fixing defects during the test development phase
3220_DLDR_fix_time	Task time applied to finding and fixing defects during the detailed design personal review phase
3220_DLDI_fix_time	Task time applied to finding and fixing defects during the detailed design inspection phase
3300_Code_fix_time	Task time applied to finding and fixing defects during the coding phase
3320_CodeR_fix_time	Task time applied to finding and fixing defects during the personal code review phase
3330_Compile_fix_time	Task time applied to finding and fixing defects during the compile phase
3340_CodeI_fix_time	Task time applied to finding and fixing defects during the code inspection phase
3350_UTest_fix_time	Task time applied to finding and fixing defects during the unit test phase
3400_TestCaseDevel_fix_time	Task time applied to finding and fixing defects during the test case development phase
4010_BITest_fix_time	Task time applied to finding and fixing defects during the build and integration testing phase
4030_STest_fix_time	Task time applied to finding and fixing defects during the system test phase
4040_Doc_fix_time	Task time applied to finding and fixing defects during the documentation phase
4050_ATest_fix_time	Task time applied to finding and fixing defects during the acceptance test phase of the project
5000_PM_fix_time	Task time applied to finding and fixing defects during the post mortem phase of the project
6100_PLife_fix_time	Task time applied to finding and fixing defects during the product life phase of the project
6200_AfterDev_fix_time	Task time applied to finding and fixing defects after the product has been developed but before the product has been released
0000_BeforeDev_def_rem	Task time applied to finding and fixing defects before the project begins

4 Results

This section provides performance results for organizations A, B, and C. It includes our findings about effectiveness and cost for each organization and answers these primary research questions:

1. How do static analysis tools affect defect escapes?
2. Does employing the static analysis tools increase or reduce total development effort?

We operationalize the first question in terms of defect finds and escapes. The second question is addressed by observing phase effort, phase defect removals, and effort required to mitigate the defects found. We use the model to estimate the secondary effects downstream of the actual static analysis.

To address these primary research questions, we must also answer these secondary questions:

1. How are the static analysis tools included in the overall development process?
2. What are the phase defect removal yields?
3. What are the find and fix times for the removed defects in each phase?
4. What are the completion rates for each phase with and without rework?

These secondary questions help us to populate the local model. We present distributions with parametric and non-parametric statistics for these values using the project as the unit of analysis.

4.1 Organization A

Organization A projects employed the static analysis tool, Tool A, as part of their personal review phase. The count of removed defects by origin phase is shown in Figure 1; the removal phase is shown in Figure 2. The injection removal count matrix is shown in Table 12. The fix time matrix is shown in Table 13, and the average fix time per defect is shown in Table 14. Summary statistics are included in Table 15.

These projects were executed in sequence by essentially the same team over a period of several years. To understand the project development parameters and consistency over time, we examined the phased effort and defects in each of the projects and as a group. We began by comparing the distribution of all defect find and fix times by project in Figure 68, Figure 69, Figure 70, and Figure 72. A visual inspection suggests that the defect fix time distribution was similar across the projects.

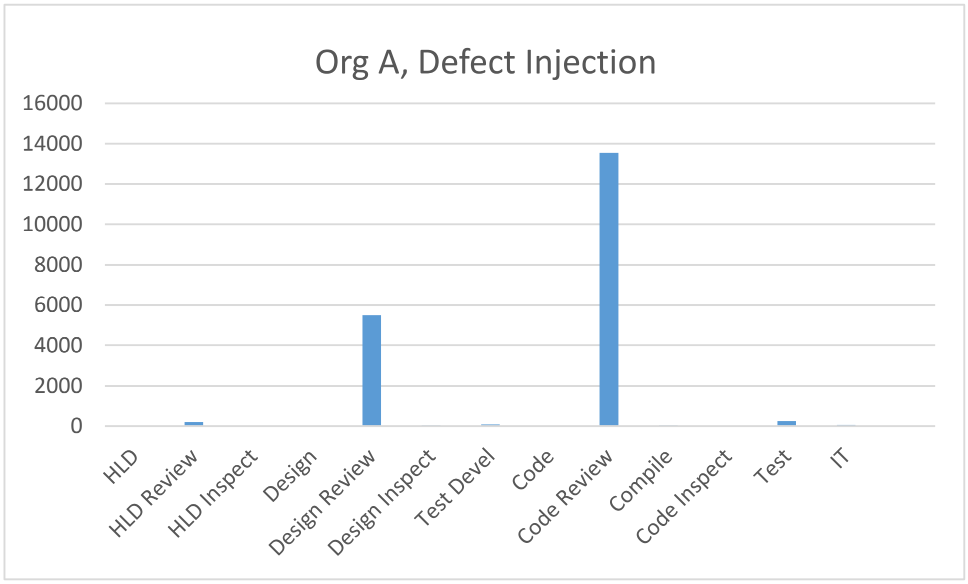


Figure 1: Defect Origin Phase (Organization A)

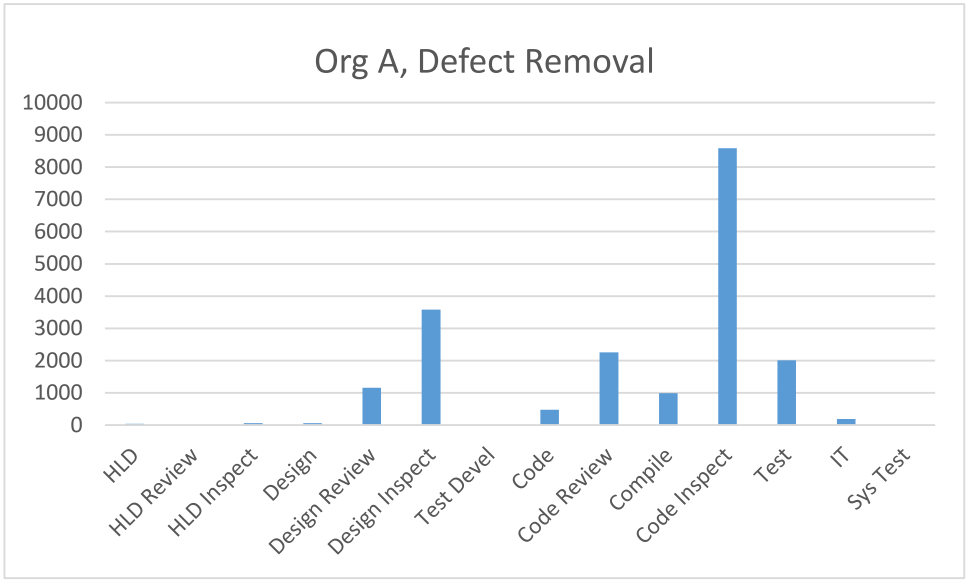


Figure 2: Defect Removal Phase (Organization A)

Table 12: Defect Count by Origin and Removal (Organization A)

	HLD	HLD Review	HLD Inspect	Design	Design Review	Design Inspect	Test Devel	Code	Code Review	Compile	Code Inspect	Test	IT	Sys Test
HLD	30	13	55	25	3	27	1	12	3		12	6		
HLD Review			2											
HLD Inspect			1											
Design				30	1118	3153	15	229	50	8	262	524		
Design Review					26	21			1					
Design Inspect						79		3	2		3			
Test Devel								34			4	2		
Code						233		141	2135	746	7722	1179		
Code Review									25	3	14	6		
Compile										12	2	2		
Code Inspect											199	2		
Test												58		1
IT													28	

Table 13: Sum of Defect Fix Time by Origin and Removal (Organization A)

	HLD	HLD Review	HLD Inspect	Design	Design Review	Design Inspect	Test Devel	Code	Code Review	Compile	Code Inspect	Test	IT	Sys Test
HLD	656.4	57.5	412.7	639.5	46.8	110.4	35.3	452.1	283.9		557.4	241.1		
HLD Review			17.0											
HLD Inspect			2.7											
Design				592.3	6867.9	16797.2	413.2	3524.6	725.8	110.5	4680.0	19493.4		
Design Review					115.0	185.1			0.4					
Design Inspect						1242.9		110.4	6.6		127.1			
Test Devel								553.6			8.5	23.6		
Code						1727.2		2001.7	10567.4	2535.1	52715.6	25218.5		
Code Review									166.5	2.6	73.5	63.5		
Compile										44.9	123.6	96.9		
Code Inspect											2013.5	54.0		
Test												1712.6		236.5
IT													1097	

Table 14: Average Defect Fix Effort by Origin and Removal (Organization A)

	HLD	HLD Review	HLD Inspect	Design	Design Review	Design Inspect	Test Devel	Code	Code Review	Compile	Code Inspect	Test	IT	Sys Test
HLD	21.9	4.4	7.5	25.6	15.6	4.1	35.3	37.7	94.6		46.4	40.2		
HLD Review			8.5											
HLD Inspect			2.7											
Design				19.7	6.1	5.3	27.5	15.4	14.5	13.8	17.9	37.2		
Design Review					4.4	8.8			0.4					
Design Inspect						15.7		36.8	3.3		42.4			
Test Devel								16.3			2.1	11.8		
Code						7.4		14.2	4.9	3.4	6.8	21.4		
Code Review									6.7	0.9	5.2	10.6		
Compile										3.7	61.8	48.5		
Code Inspect											10.1	27.0		
Test												29.5		236.5
IT													39.2	

Table 15: Phase Defect Fix Effort Statistics (Organization A)

Removed_Phase	N	Mean	SE Mean	StDev	Minimum	Q1	Med	Q3	Max	IQR
HLD	15	67.7	16.7	64.8	1.8	18.1	39.7	120.0	195.0	101.9
HLD Review	11	5.2	2.1	6.9	0.1	0.5	3.7	5.4	20.8	4.9
HLD Inspect	51	8.5	2.0	14.5	0.0	1.4	2.8	10.1	76.0	8.7
Design	56	22.2	5.3	40.0	0.0	2.9	6.1	24.1	241.5	21.3
Design Review	814	9.0	0.8	23.3	0.0	1.0	3.0	8.0	364.5	7.0
Design Inspect	2296	9.1	0.5	22.4	0.0	1.1	3.0	7.9	367.1	6.8
Code	373	24.5	3.8	73.9	0.0	2.0	5.5	16.7	992.0	14.7
Code Review	1592	7.6	0.5	19.5	0.0	0.9	2.1	6.0	270.0	5.1
Compile	293	10.3	2.4	40.6	0.1	0.9	2.5	6.2	603.0	5.3
Code Inspect	6571	10.1	0.3	27.1	0.0	1.2	3.4	8.6	640.5	7.4
IT	127	42.3	10.0	112.3	0.0	2.5	10.4	38.5	1060.0	36.0
After Development	4	43.7	21.6	43.2	3.6	5.3	40.6	85.2	90.0	79.9

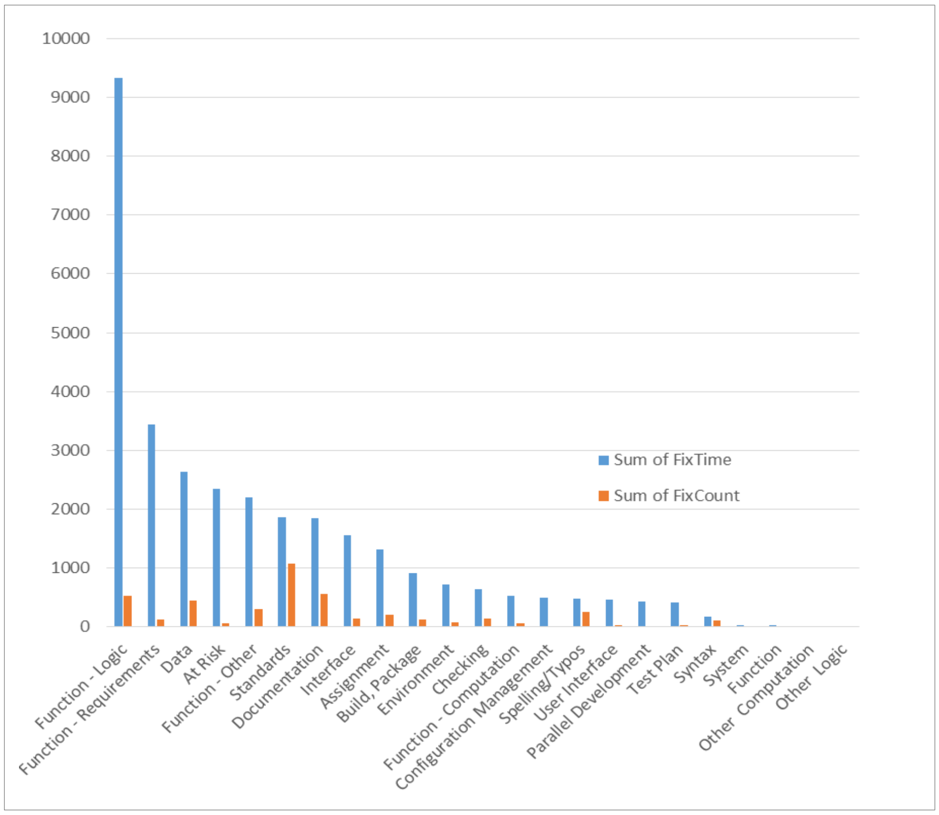


Figure 3: Defect Counts and Fix Time by Type (Organization A)

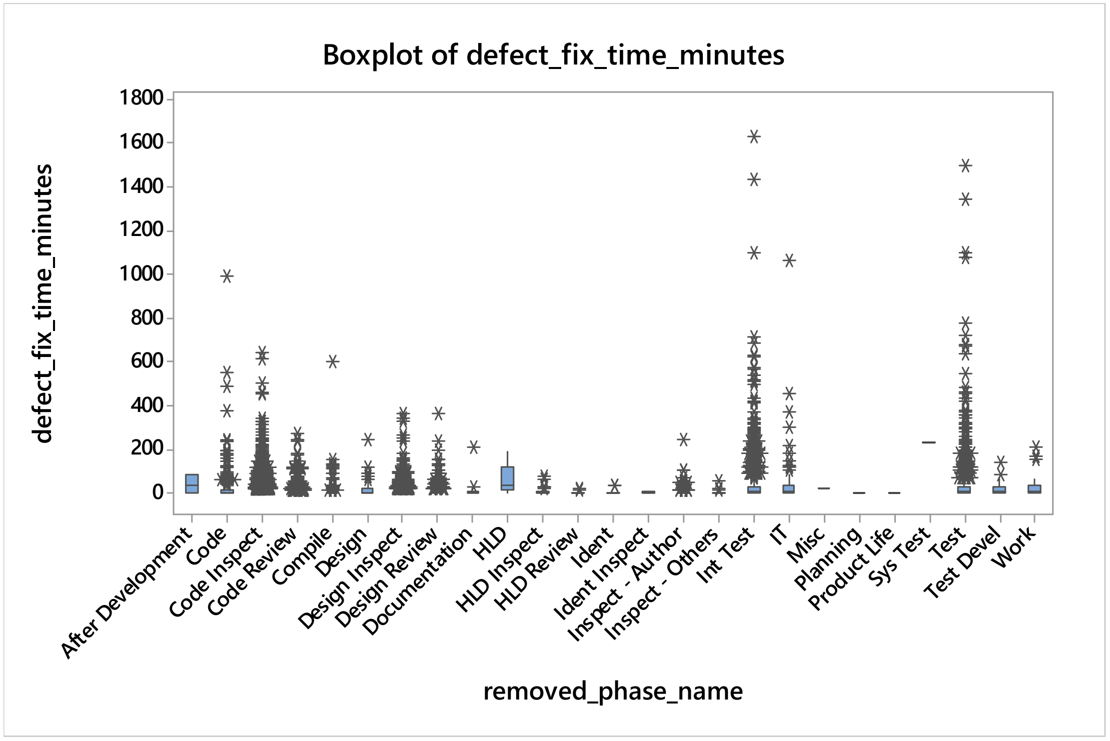


Figure 4: Phase Defect Fix Effort Box Plot (Organization A)

Table 16: Cost and Quality Performance in the Absence of Tool_A

Phase	No.Defect.Phase. Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/ Defect]
0000_BeforeDev	0.00000		0.00	
1000_Misc	0.01923	0.000		0.417
1100_Strat	0.00018	0.000		
1150_Planing	0.00068	0.016		0.012
1200_SE_REQ	0.02440	0.048		0.276
1220_SE_REQR	0.00434	0.028	0.25	0.066
1240_SE_REQI	0.00321	0.027		0.051
1250_SE_REQ_val	0.00443	0.010	0.12	0.477
3000_Req	0.00000			
3020_ReqR	0.00005	0.062		
3040_ReqI	0.00408	0.001		
3100_HLD	0.00410	0.077		0.458
3110_ITP	0.00000			
3120_HLDR	0.00000	2.087	0.01	0.074
3140_HLDI	0.00122	0.001	0.04	0.124
3200_DLD	0.00691	1.234		0.351
3210_TD	0.00023	0.259		0.467
3220_DLDR	0.00174	0.039	0.16	0.105
3220_DLDI	0.00333	0.035	0.60	0.097
3300_Code	0.00945	2.174		0.318
3320_CodeR	0.00141	0.044	0.14	0.090
3330_Compile	0.00036	0.061	0.07	0.051
3340_CodeI	0.00468	0.062	0.68	0.128
3350_UTest	0.00561	0.015	0.51	0.441
3551xxxx	0.00000	0.000	0.00	0.000
3400_TestCaseDevel	0.00000			
4010_BITest	0.00300	0.051	0.40	0.508
4015_xxxx	0.00000	0.000	0.00	0.000
4030_STest	0.00000	0.000	0.17	3.942
4040_Doc	0.00233	0.035	1.00	0.148
4050_ATest	0.00000		0.00	
5000_PM	0.00039	0.000		
6100_PLife	0.00000	23.077	0.00	0.043

To simulate the cost and quality performance of Organization A projects in the absence of static analysis, the data was modified in the following ways:

- The personal review phase (i.e., 3320_CodeR) was removed (i.e., variable values were set to zero).
- The documentation phase was removed from consideration (since documentation activities do not affect software code defect values). The yield was changed from 1.00% to 0.00%.

Since a small number of defects escaped into the system test phase, the yield of 0.17% for this phase was not modified within the cost quality model for simulating the case where a static analysis tool was *not* used.

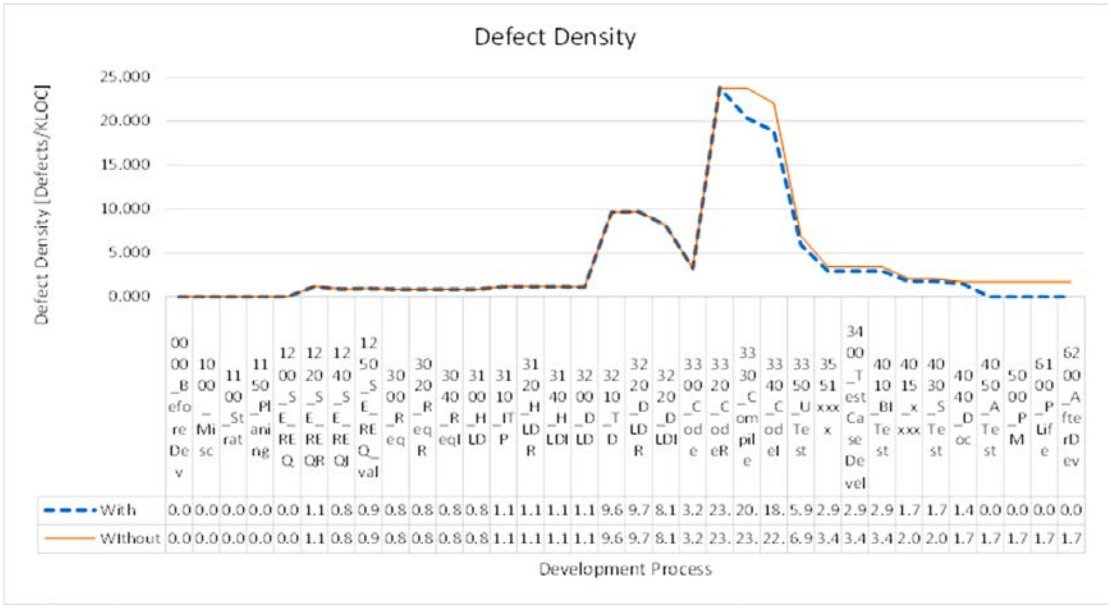
Table 17 presents cost and quality performance data for Organization A in the absence of Tool A during the personal review phase.

Table 17: Cost and Quality Performance in the Absence of Tool_A (Organization A)

Phase	No.Defect.Phase. Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
0000_BeforeDev	0.00000		0.00	
1000_Misc	0.01923	0.000		0.417
1100_Strat	0.00018	0.000		
1150_Planing	0.00068	0.016		0.012
1200_SE_REQ	0.02440	0.048		0.276
1220_SE_REQR	0.00434	0.028	0.25	0.066
1240_SE_REQI	0.00321	0.027		0.051
1250_SE_REQ_val	0.00443	0.010	0.12	0.477
3000_Req	0.00000			
3020_ReqR	0.00005	0.062		
3040_ReqI	0.00408	0.001		
3100_HLD	0.00410	0.077		0.458
3110_ITP	0.00000			
3120_HLDR	0.00000	2.087	0.01	0.074
3140_HLDI	0.00122	0.001	0.04	0.124
3200_DLD	0.00691	1.234		0.351
3210_TD	0.00023	0.259		0.467
3220_DLDR	0.00174	0.039	0.16	0.105
3220_DLDI	0.00333	0.035	0.60	0.097
3300_Code	0.00945	2.174		0.318
3320_CodeR	0.00000	0.000	0.00	0.000
3330_Compile	0.00036	0.061	0.07	0.051
3340_CodeI	0.00468	0.062	0.68	0.128

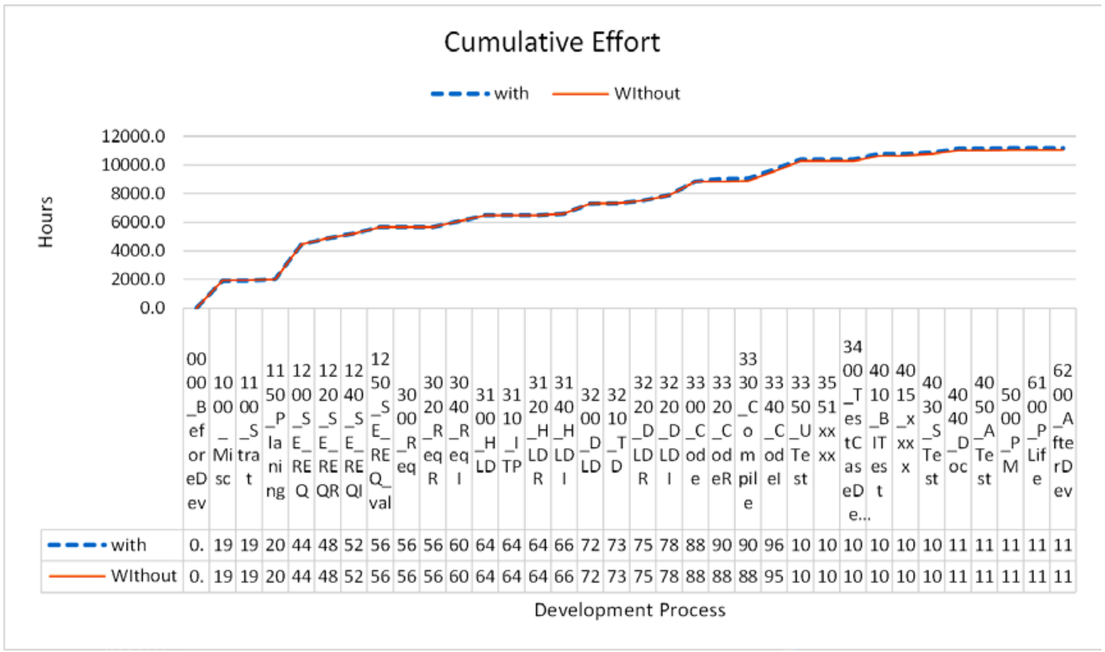
Phase	No.De-fect.Phase. Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
3350_UTest	0.00561	0.015	0.51	0.441
3551xxxx	0.00000	0.000	0.00	0.000
3400_Test-CaseDevel	0.00000			
4010_BITest	0.00300	0.051	0.40	0.508
4015_xxxx	0.00000	0.000	0.00	0.000
4030_STest	0.00000	0.000	0.17	3.942
4040_Doc	0.00233	0.035	0.00	0.148
4050_ATest	0.00000		0.00	
5000_PM	0.00039	0.000		
6100_PLife	0.00000	23.077	0.00	0.043

Figure 5 illustrates the impact on defect density from removing the personal review process phase (which included Static Analysis Tool A) from the development.



Development Process	With	Without
0000_BeforeDev	0.0	0.0
1000_Misc	0.0	0.0
1100_Strat	0.0	0.0
1150_Planing	0.0	0.0
1200_SE_REQ	0.0	0.0
1220_SE_REQR	1.1	1.1
1240_SE_REQI	0.8	0.8
1250_SE_REQ_val	0.9	0.9
3000_Req	0.8	0.8
3020_ReqR	0.8	0.8
3040_ReqI	0.8	0.8
3100_HLD	0.8	0.8
3110_ITP	1.1	1.1
3120_HLDR	1.1	1.1
3140_HLDI	1.1	1.1
3200_DLD	1.1	1.1
3210_TD	9.6	9.6
3220_DLDR	9.7	9.7
3220_DLDI	8.1	8.1
3300_Code	3.2	3.2
3320_CodeR	23.	23.
3330_Compile	20.	23.
3340_CodeI	18.	22.
3350_UTest	5.9	6.9
3551xxxx	2.9	3.4
3400_TestCaseDevel	2.9	3.4
4010_BITest	2.9	3.4
4015_xxxx	1.7	2.0
4030_STest	1.7	2.0
4040_Doc	1.4	1.7
4050_ATest	0.0	1.7
5000_PM	0.0	1.7
6100_PLife	0.0	1.7
6200_AfterDev	0.0	1.7

Figure 5: Defect Density with and without Static Analysis Tool A (Organization A)



	0000_BeforeDev	1000_Misc	1100_Strat	1150_Planing	1200_SE_REQ	1220_SE_REQR	1240_SE_REQI	1250_SE_REQ_val	3000_Req	3020_ReqR	3040_ReqI	3100_HLD	3110_ITP	3120_HLDR	3140_HLDI	3200_DLD	3210_TD	3220_DLDR	3220_DLDI	3300_Code	3320_CodeR	3330_Compile	3340_CodeI	3350_UTest	3551xxxx	3400_TestCaseDe...	4010_BITest	4015_xxxx	4030_STest	4040_Doc	4050_ATest	5000_PM	6100_PLife	6200_AfterDev
with	0.	19	19	20	44	48	52	56	56	56	60	64	64	64	66	72	73	75	78	88	90	90	96	10	10	10	10	10	10	11	11	11	11	11
WIthout	0.	19	19	20	44	48	52	56	56	56	60	64	64	64	66	72	73	75	78	88	88	88	95	10	10	10	10	10	10	11	11	11	11	11

Figure 6: Cumulative Development Effort with and without Static Analysis Tool A (Organization A)

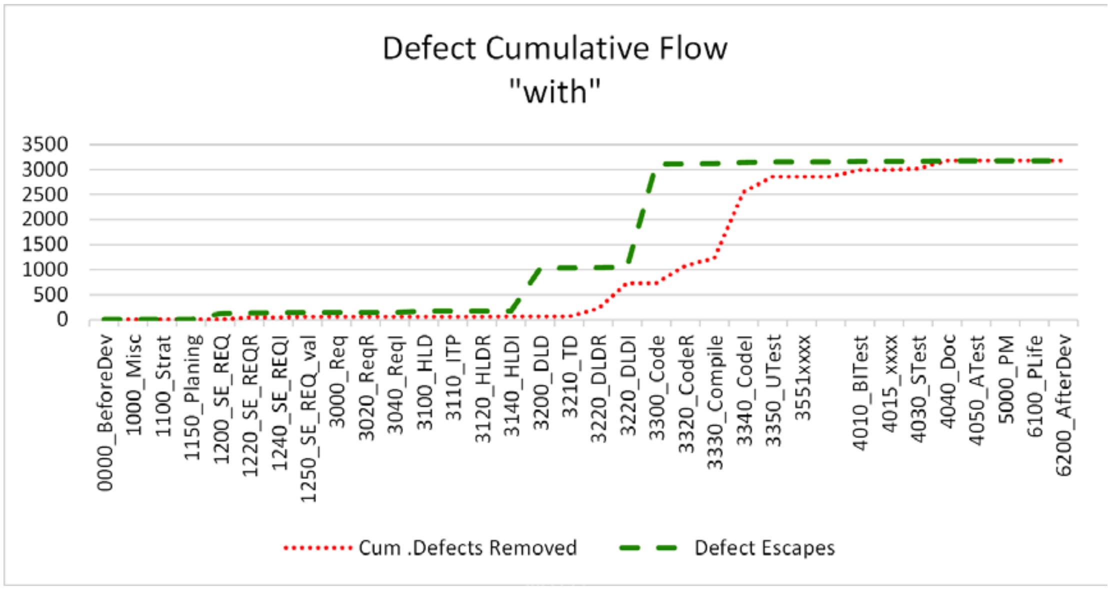


Figure 7: Cumulative Defect Flow with Static Analysis Tool A (Organization A)

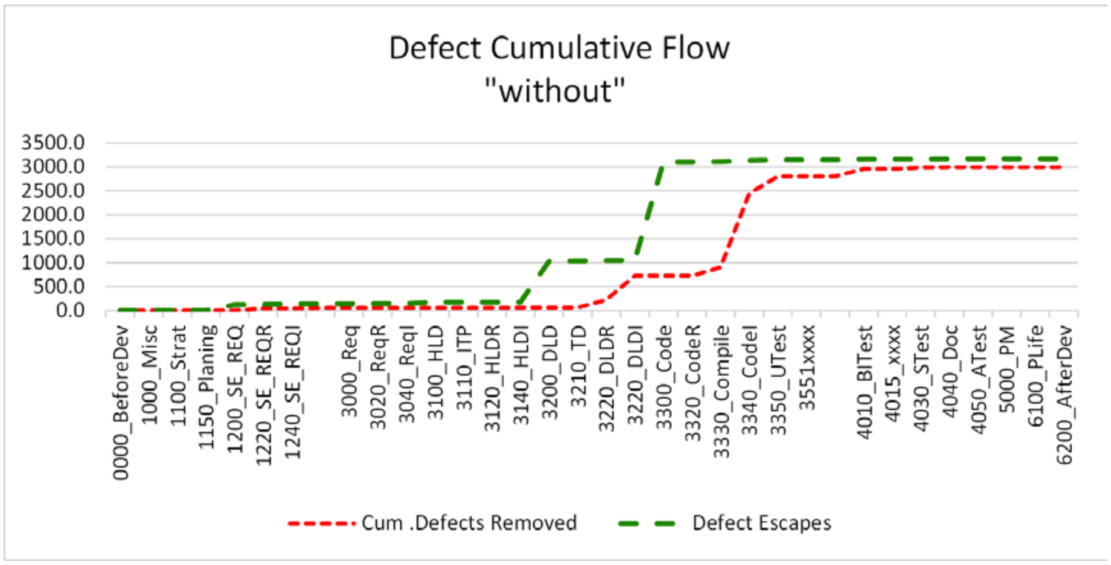


Figure 8: Cumulative Defect Flow without Static Analysis Tool A (Organization A)

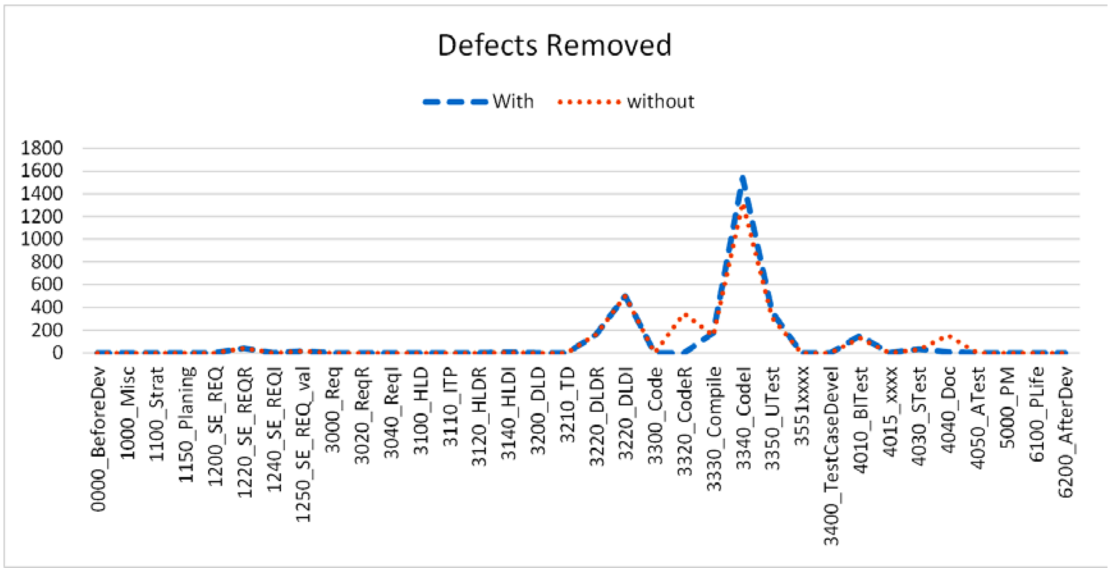


Figure 9: Defect Removal by Phase with and without Static Analysis (Organization A)

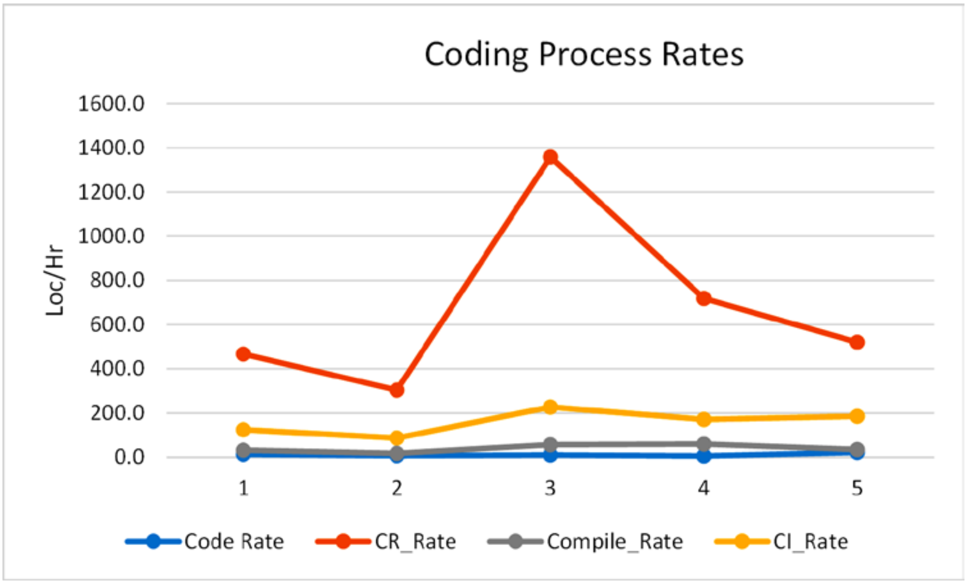


Figure 10: Coding Process Rates (Organization A)

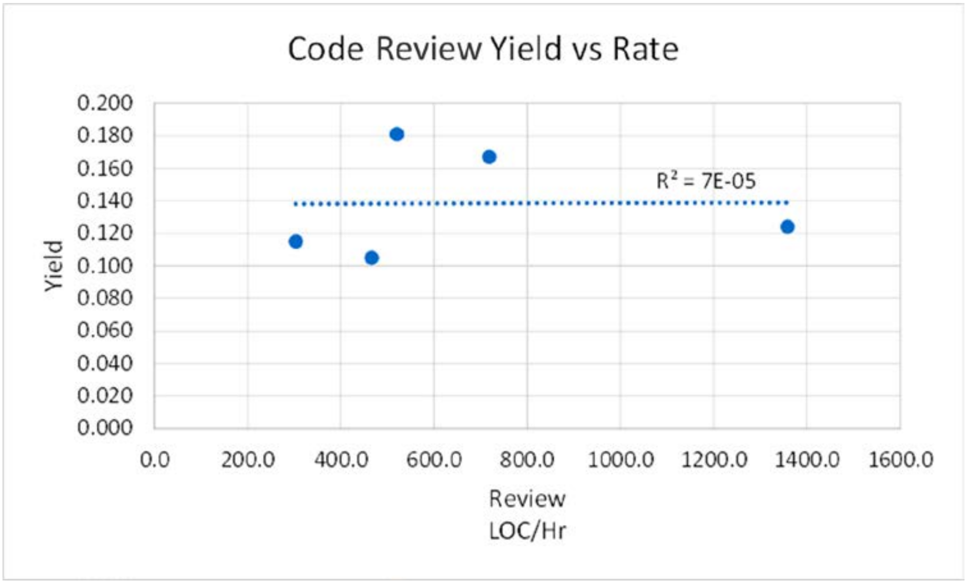


Figure 11: Code Review Yield vs. Review Rate (Organization A)

4.2 Organization B

Organization B projects employed the static analysis tools Tool_B_1 and Tool_B_2. A histogram of defect removals is shown in Figure 13. The tools were primarily used in the compile phase of development, and then in the personal review phase and the inspection phase. The counts of discovered defect types by orthogonal defect category are shown in Figure 12. The most common types are violations of development standards and inconsistent interfaces.

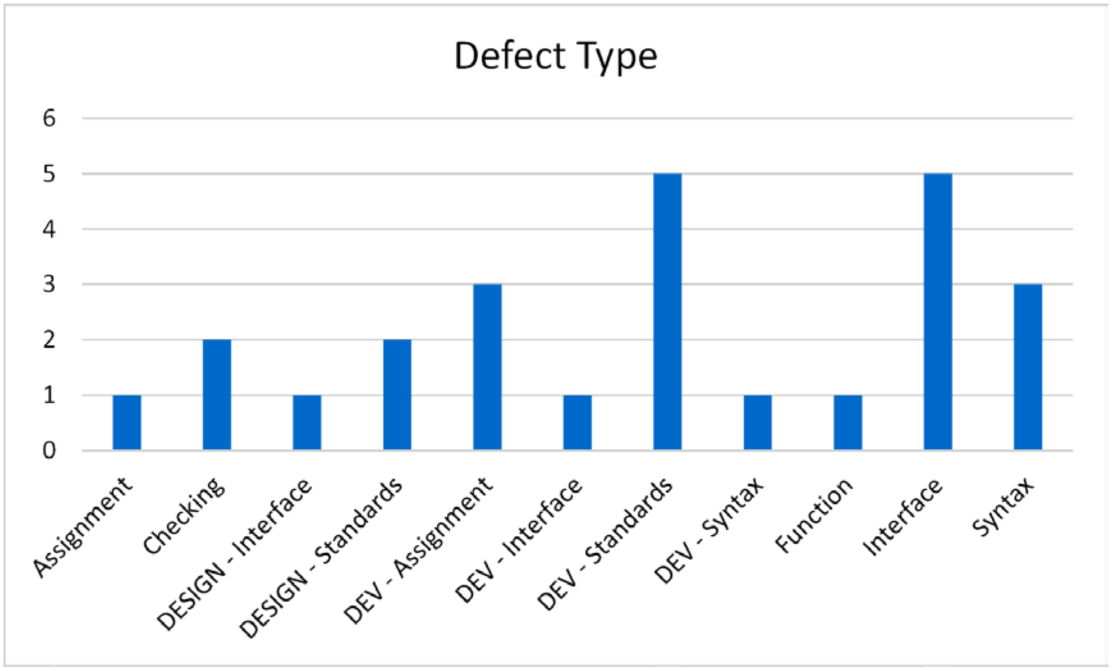


Figure 12: Defect Types (Organization B)

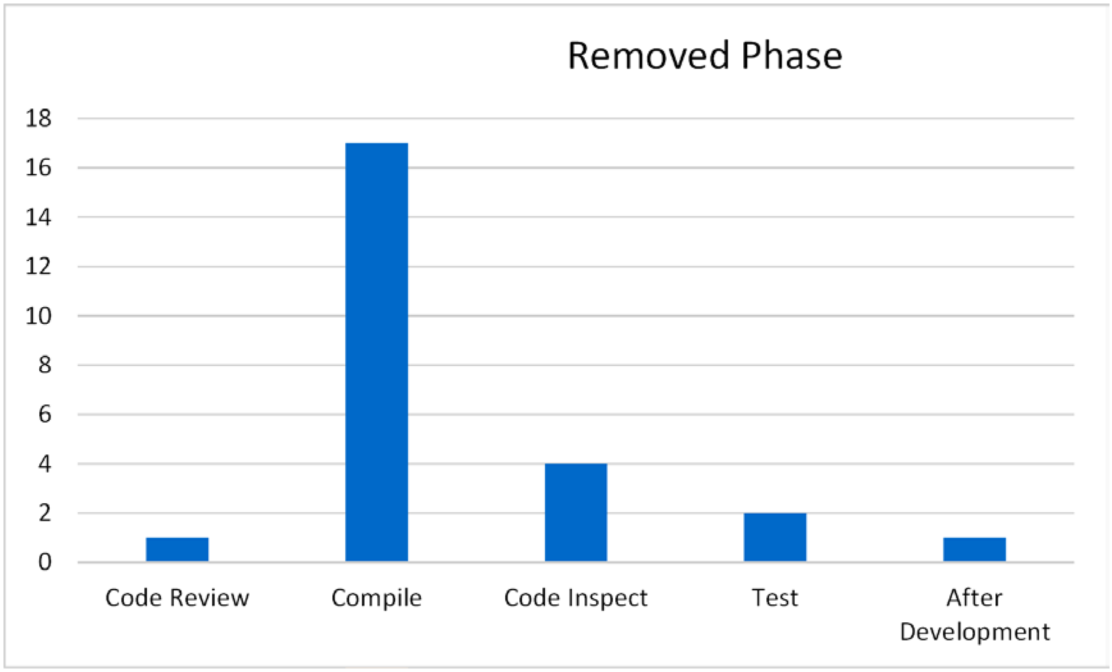


Figure 13: Number of Defects Removed During Development Phases (Organization B)

In order to address the main research question (*What are the model parameters with and without using these tools?*) we performed additional analysis on the defect rates and fix times. The related research questions included the following:

1. Where were the defects injected?
2. Where were the defects removed?
3. How much effort was spent per defect by phase?
4. How much total effort was required to use the tools?

Tool_B_1 defect finds by phase origin are presented in Table 18. Defect injections are shown along the rows, and defect removals are shown in the columns. Table cells exclude the phases prior to design. Defect finds were more or less equal in the code review, code inspect, and compile phases. The effort recorded fixing these defects is summarized in Table 19. The average find and fix times by removal and origin phase are summarized in Table 20. Similar tables for Tool_B_2 defects are summarized in Table 22, Table 23, and Table 24.

The descriptive statistics for the removal phase are summarized in Table 27. Detailed parametric and non-parametric descriptions of find and fix time in each phase are included in Figure 14, Figure 15, Figure 19, Figure 20, and Figure 21.

Table 18: Tool_B_1 Origin and Removal Phase of Defect Found

Sum of Fix Count	**Removed**						
Injected	Design Inspect	Code	Code Review	Code Inspect	Compile	After Devel-opment	Grand Total
Before Development		1		1	1		3
Design	2						2
Code			10	14	10		34
Compile				1			1
Code Inspect				3			3
Test						1	1
Grand Total	2	1	10	19	11	1	44

Table 19: Tool_B_1 Total Defect Fix Effort by Phase of Origin and Removal

Sum of Tool_B_1-Effort	**Removed**							
Injected	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	After Devel-opment	Grand Total
Before Development		260.4		30.5	1			291.9
Design	0.8			0	0			0.8
Code			48.2	89.1	115.1			252.4
Code Inspect					30.6	0		30.6
Compile					4.1			4.1
Test							3.3	3.3
Grand Total	0.8	260.4	48.2	119.6	150.8	0	3.3	583.1

Table 20: Tool_B_1 Fix Effort per Defect by Phase of Origin and Removal

Average Fix Effort per Defect	**Removed**							
Injected	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	After Development	Grand Total
Before Development		86.80		6.10	1.00			32.43
Design	0.40							0.40
Code			4.82	8.91	8.85			7.65
Code Inspect					10.20			10.20
Compile					4.10			4.10
Test							3.30	3.30
Grand Total	0.4	86.8	4.82	7.97	8.38		3.3	11.9

These are a small portion of all development defects. Table 24 summarizes counts of all defects found and removed by phase. Total and average fix times by phase origin and removal are included in Table 25 and Table 26. Descriptive statistics for all defects are summarized in Table 27. Defect types are included in Table 28. Histograms of the fix time distributions, along with statistics are included in Figure 14 through Figure 22. Category_0 refers to Tool_B_1, which only scans source code, while Category_1 refers to Tool_B_2, which also scans the compiled binary.

The graphic data provides a way to visualize the statistical significance of the differences in the mean values for different phases or removal activities. The range of find and fix times is wide and the distributions are highly skewed, but the distributions are unimodal. Because our model will add data between phases rather than multiply, use the mean values to obtain average behavior. Nonetheless, distribution data suggests that we can expect wide ranges of behavior with small samples. Future work may use Monte Carlo rather than point solutions to obtain expected performance ranges.

Table 21: Defects Coded for Tool_B_2 by Phase of Origin and Removal

Tool_B_2	**Removed**							
Injected	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	After Development	Grand Total
Before Development								
Design				5	1			6
Code			1	12	1			14
Code Inspect					3	2		5
Compile								
Test							1	1
Grand Total			1	17	5	2	1	26

Table 22: Tool_B_2 Defect Removal Effort by Phase of Origin and Removal

Sum of FX-Effort	**Removed**							
Injected	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	After Development	Grand Total
Before Development		0		0	0			0
Design	0			8.3	3.3			11.6
Code			1.7	20.8	0.5			23
Code Inspect					34.7	31		65.7
Compile					0			0
Test							3.3	3.3
Grand Total	0	0	1.7	29.1	38.5	31	3.3	103.6

Table 23: Tool_B_2 Removal Effort per Defect by Phase of Origin and Removal

Average Fix Effort per Defect	**Removed**							
Injected	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	After Development	Grand Total
Before Development								
Design				1.66	3.30			1.93
Code			1.70	1.73	0.50			1.64
Code Inspect					11.57	15.50		13.14
Compile								
Test							3.30	3.30
Grand Total			1.7	1.71	7.7	15.5	3.3	3.98

Table 24: All Defects, Phase of Origin (Injection) and Removal

Defects Removed													
Phase Injected	Design Review	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	1070	1607	271	85	19	220	246	2	20			7	3547
Design Review	17	6	2	1		1	1						28
Design Inspect		48	45	13	1	20	7		1				135
Code			6	1204	361	2160	230	11	27		1	2	4002
Code Review				8	14	4	6						32
Code Inspect						84	65	1	1				151
Compile						10	1						11
Test Devel			8	6		4	82	128	298				526
Test							19	4	15			1	39
Int Test								7	34	5			46
Grand Total	1087	1661	332	1317	395	2503	657	153	396	5	1	10	8517

Table 25: Total Defect Find and Fix Time (After High-Level Design)

Sum of Fix Time [Minutes]													
Phase Injected	Design Review	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	6172.8	10798.3	2378.2	1008.9	150.9	1981.2	6986.6	30.9	1120.7			830.2	31458.7
Design Review	198.7	23.1	2.9	21.8		2.5	23.7						272.7
Design Inspect		209.6	141.6	124.2	1.2	129	30.5		14.4				650.5
Code			23.3	5169.4	1091.1	10868.7	2507.8	586.9	984.3		99.8	56.8	21388.1
Code Review				33.4	23.9	34.6	30.7						122.6
Code Inspect						455.5	604.9	182	11				1253.4
Compile						36.6	11						47.6
Test Devel			51.8	42.3		129.8	127.4	347.8	752.7				1451.8
Test							212.8	1.7	82.6			3.3	300.4
Int Test								9.4	151.4	4.1			164.9
Grand Total	6371.5	11031	2597.8	6400	1267.1	13637.9	10535.4	1158.7	3117.1	4.1	99.8	890.3	57110.7

Table 26: Average Defect Fix Effort by Removal and Origin

Average Defect Effort [Minutes/Defect]													
Phase Injected	Design Review	Design Inspect	Code	Code Review	Compile	Code In-spect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	5.8	6.7	8.8	11.9	7.9	9.0	28.4	15.5	56.0			118.6	8.87
Design Review	11.7	3.9	1.5	21.8		2.5	23.7						9.74
Design Inspect		4.4	3.1	9.6	1.2	6.5	4.4		14.4				4.82
Code			3.9	4.3	3.0	5.0	10.9	53.4	36.5		99.8	28.4	5.34
Code Review				4.2	1.7	8.7	5.1						3.83
Code Inspect						5.4	9.3	182.0	11.0				8.30
Compile						3.7	11.0						4.33
Test Devel			6.5	7.1		32.5	1.6	2.7	2.5				2.76
Test							11.2	0.4	5.5			3.3	7.70
Int Test								1.3	4.5	0.8			3.58
Total	5.86155	6.64118	7.8247	4.85953	3.20785	5.44862	16.0356	7.5732	7.87146	0.82	99.8	89.03	6.71

Table 27: Descriptive Statistics for All Defects by Removal Phase

Phase	N	Mean	SE Mean	StDev	Minimum	Q1	Median	Q3	Maximum
Fix Time Accept Test	5	0.82	0.536	1.199	0.1	0.1	0.2	1.85	2.9
After Development	10	89	24.2	76.5	3.3	24.4	72.3	142.8	240
Code	344	9.27	1.34	24.86	0	1.1	2.9	6.4	257.4
Code Inspect	2607	5.404	0.258	13.152	0	0.7	1.6	4.7	242.9
Code Review	1337	4.894	0.315	11.5	0.1	0.8	1.8	4.3	171.1
Compile	415	3.258	0.541	11.014	0.1	0.6	1.2	2.6	198.9
Design	4	13.15	4.97	9.95	3.1	4	12.55	22.9	24.4
Design Inspect	1690	6.82	0.459	18.862	0	0.7	1.9	5.125	363.3
Design Review	1094	5.961	0.506	16.743	0.1	0.8	1.75	4.4	239.7
Documentation	49	10.89	2.03	14.24	0.2	1.65	5.4	17.45	72.2
HLD Inspect	35	9.59	3.16	18.71	0.2	1.2	3.1	11.3	103.3
HLD Review	19	9.08	3.16	13.77	0.7	1.6	3.5	13	59.9
Int Test	153	7.57	3.22	39.89	0.1	0.5	1	4	459.3
Int Test Plan	79	2.172	0.577	5.126	0	0.1	0.5	2	40
Planning	7	30.9	11.2	29.8	0.7	2.3	23.9	49	82.8
Product Life	2	261	161	228	100	*	261	*	422
Reqts Inspect	469	3.861	0.327	7.074	0	0.8	2.6	5	116.6
Reqts Review	172	2.222	0.36	4.727	0.1	0.4	1	2.2	54.6
Sys Test	401	8.31	1.76	35.3	0	0.3	0.8	2.45	510
Test	691	18.27	1.68	44.26	0	2.1	5.5	14.8	654
Test Devel	6	14.32	4.77	11.69	0.5	6.65	11.6	22.88	34.5

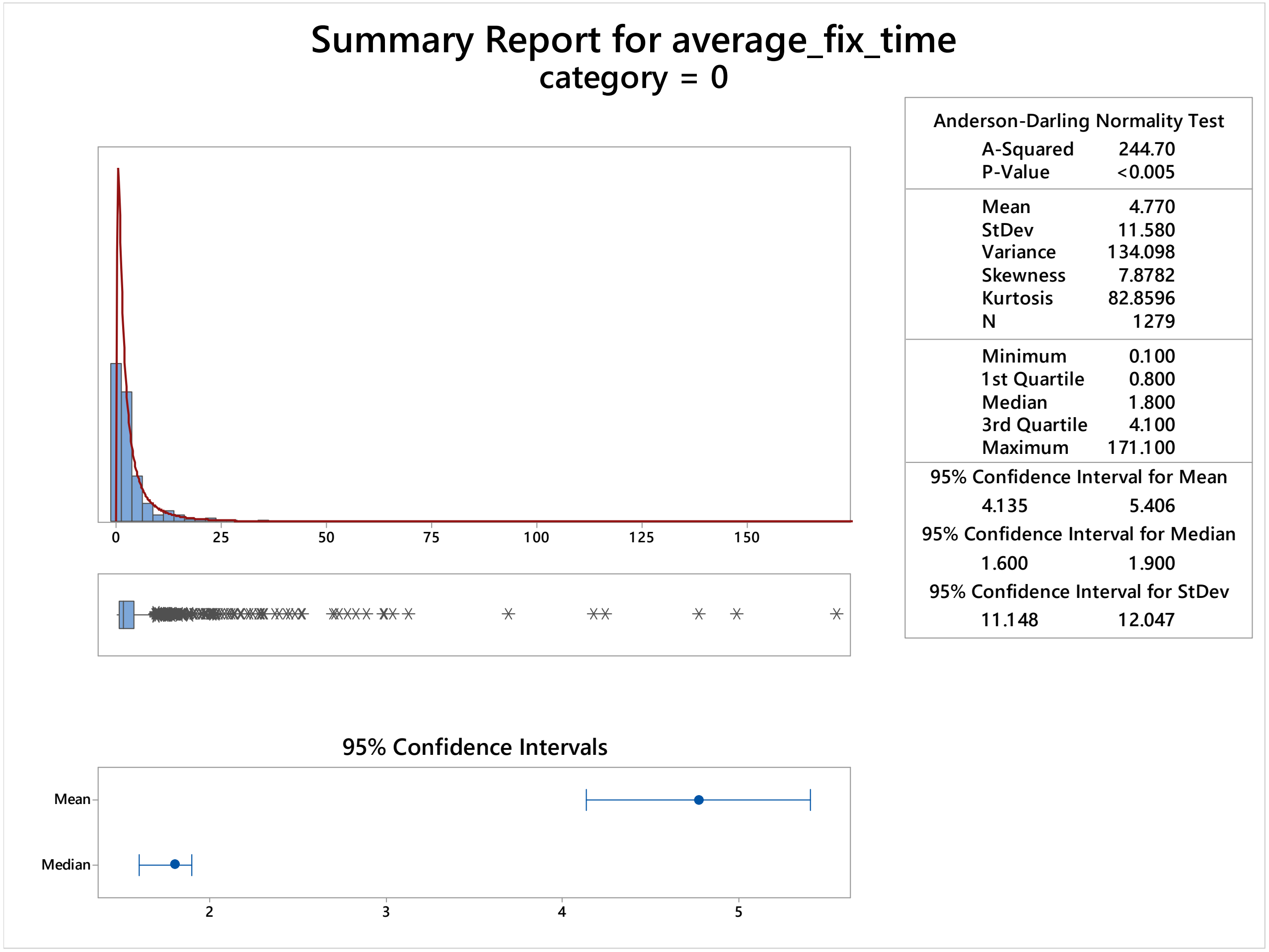
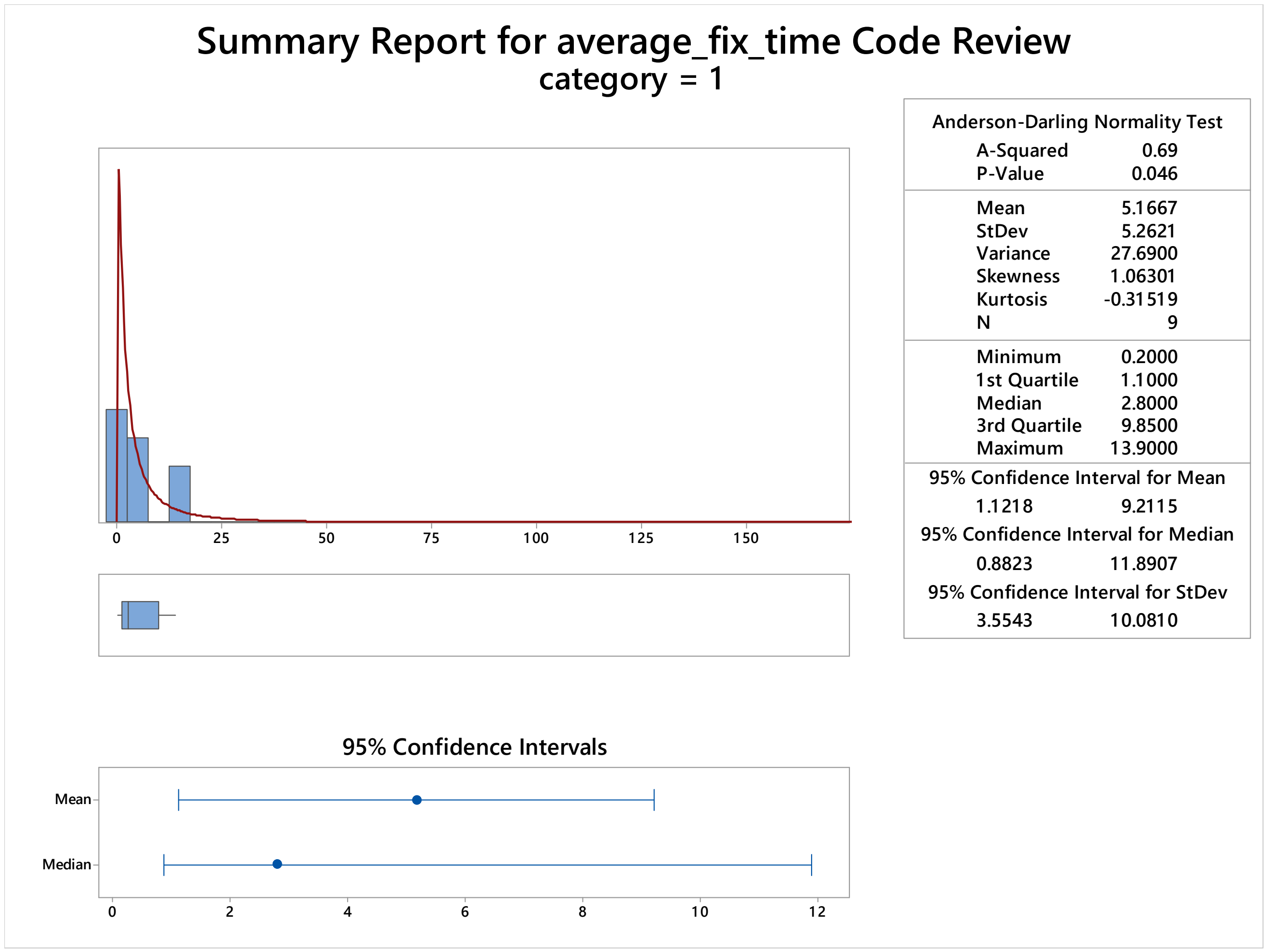
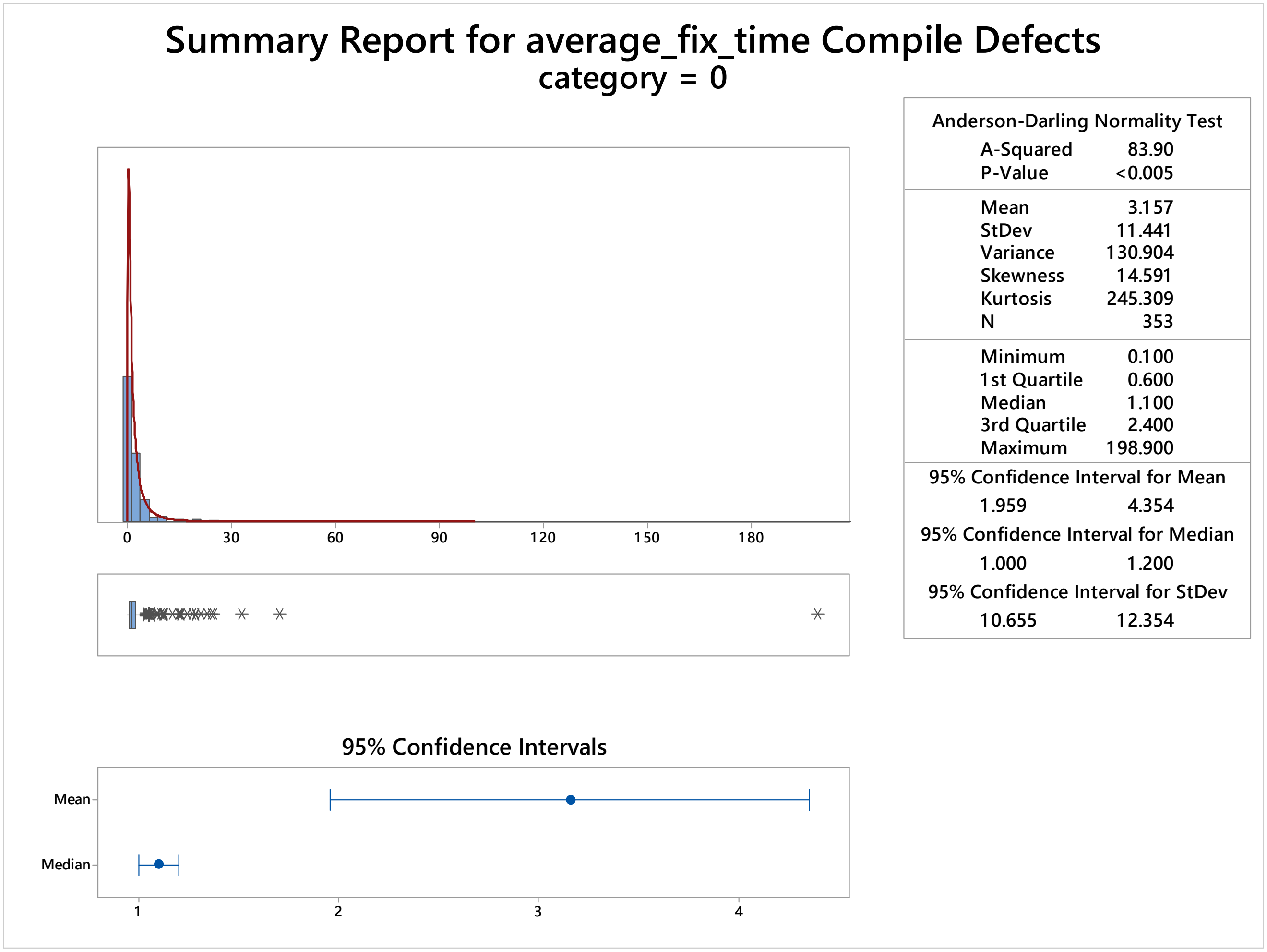


Figure 14: All Other Defects, Code Review



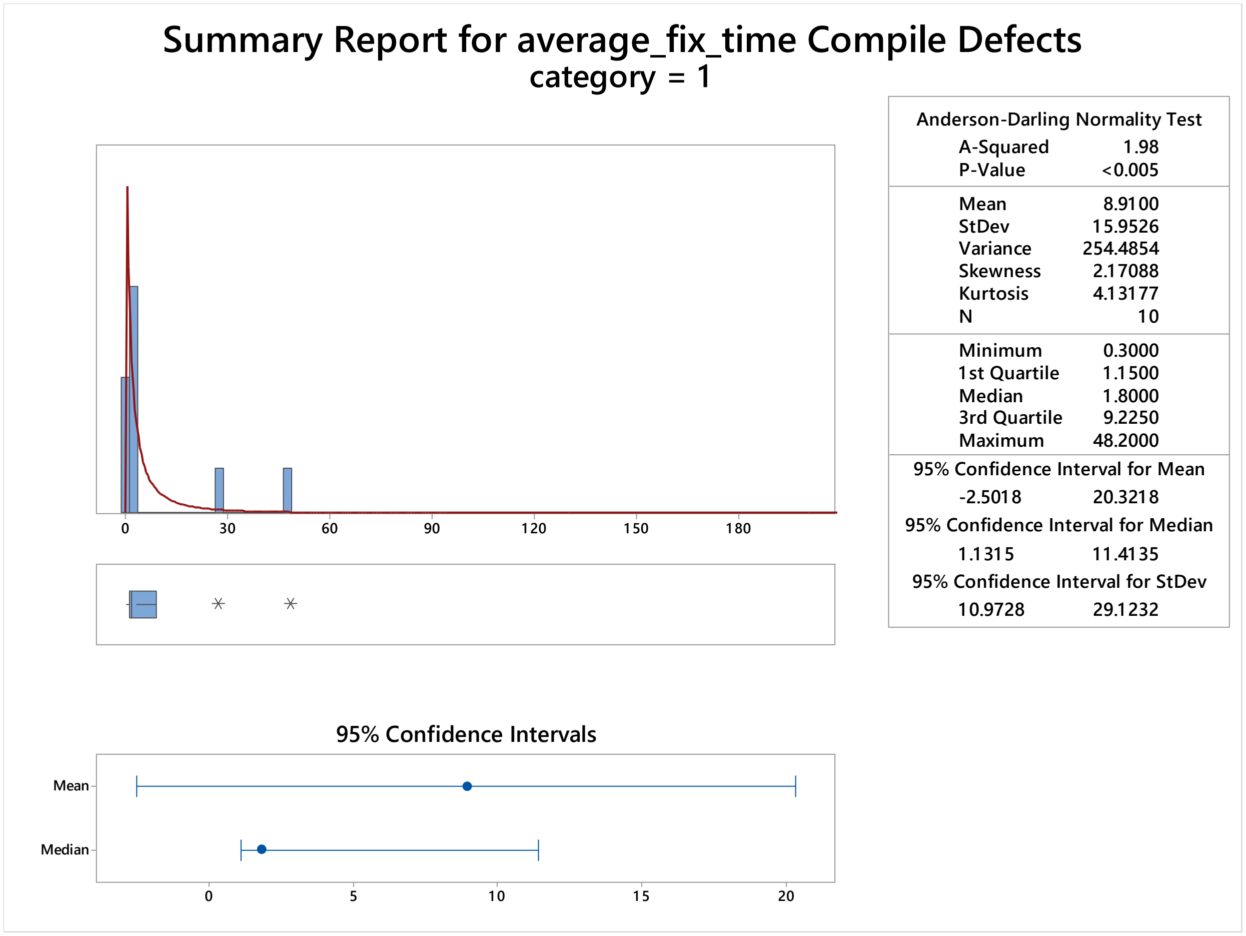
Anderson-Darling Normality Test	
A-Squared	0.69
P-Value	0.046
Mean	5.1667
StDev	5.2621
Variance	27.6900
Skewness	1.06301
Kurtosis	-0.31519
N	9
Minimum	0.2000
1st Quartile	1.1000
Median	2.8000
3rd Quartile	9.8500
Maximum	13.9000
95% Confidence Interval for Mean	
1.1218	9.2115
95% Confidence Interval for Median	
0.8823	11.8907
95% Confidence Interval for StDev	
3.5543	10.0810

Figure 15: Tool_B_1 Defect Find and Fix Time, Code Review



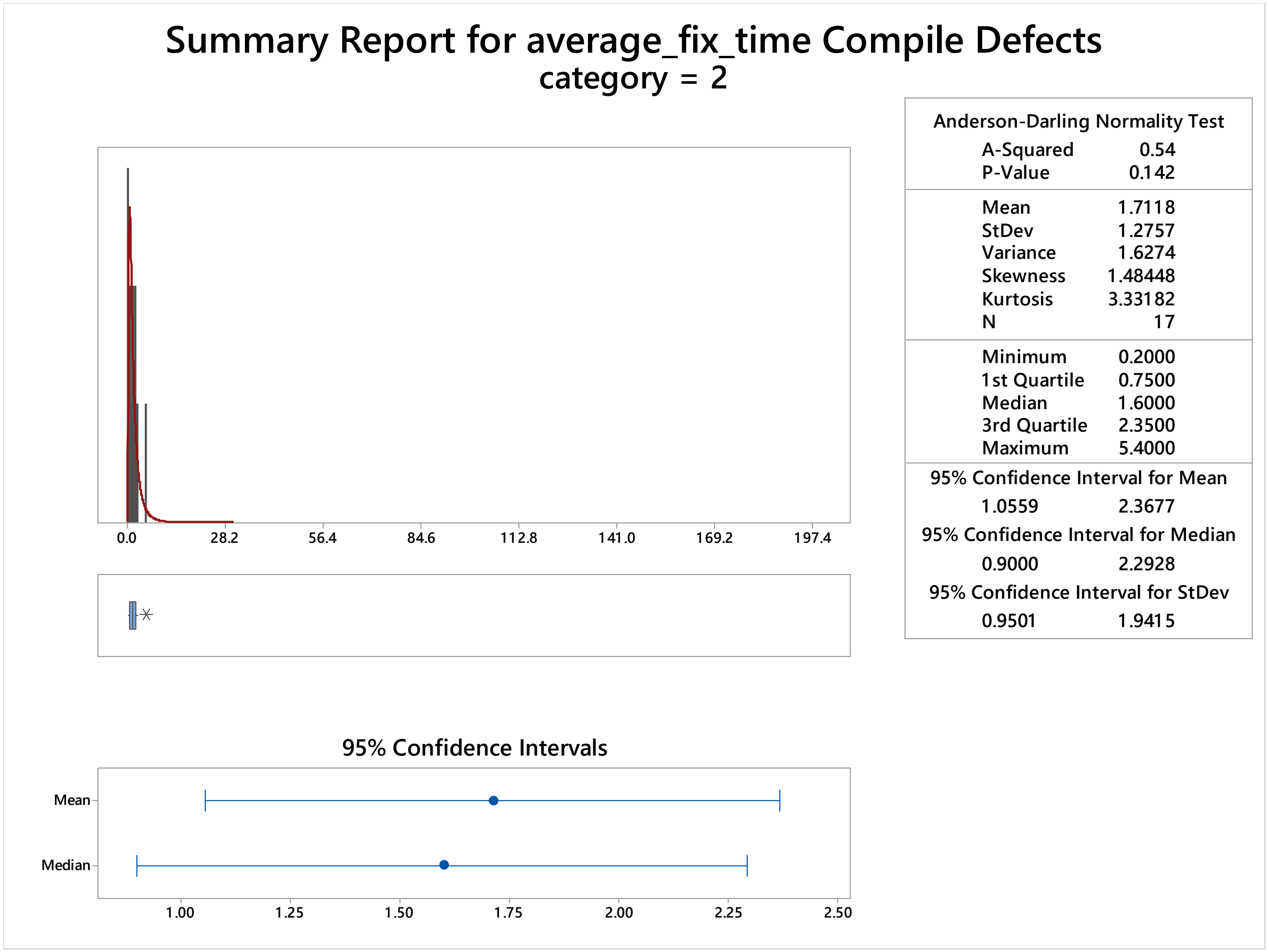
Anderson-Darling Normality Test	
A-Squared	83.90
P-Value	<0.005
Mean	3.157
StDev	11.441
Variance	130.904
Skewness	14.591
Kurtosis	245.309
N	353
Minimum	0.100
1st Quartile	0.600
Median	1.100
3rd Quartile	2.400
Maximum	198.900
95% Confidence Interval for Mean	
1.959	4.354
95% Confidence Interval for Median	
1.000	1.200
95% Confidence Interval for StDev	
10.655	12.354

Figure 16: Other Defects, Average Find and Fix Time, Compile



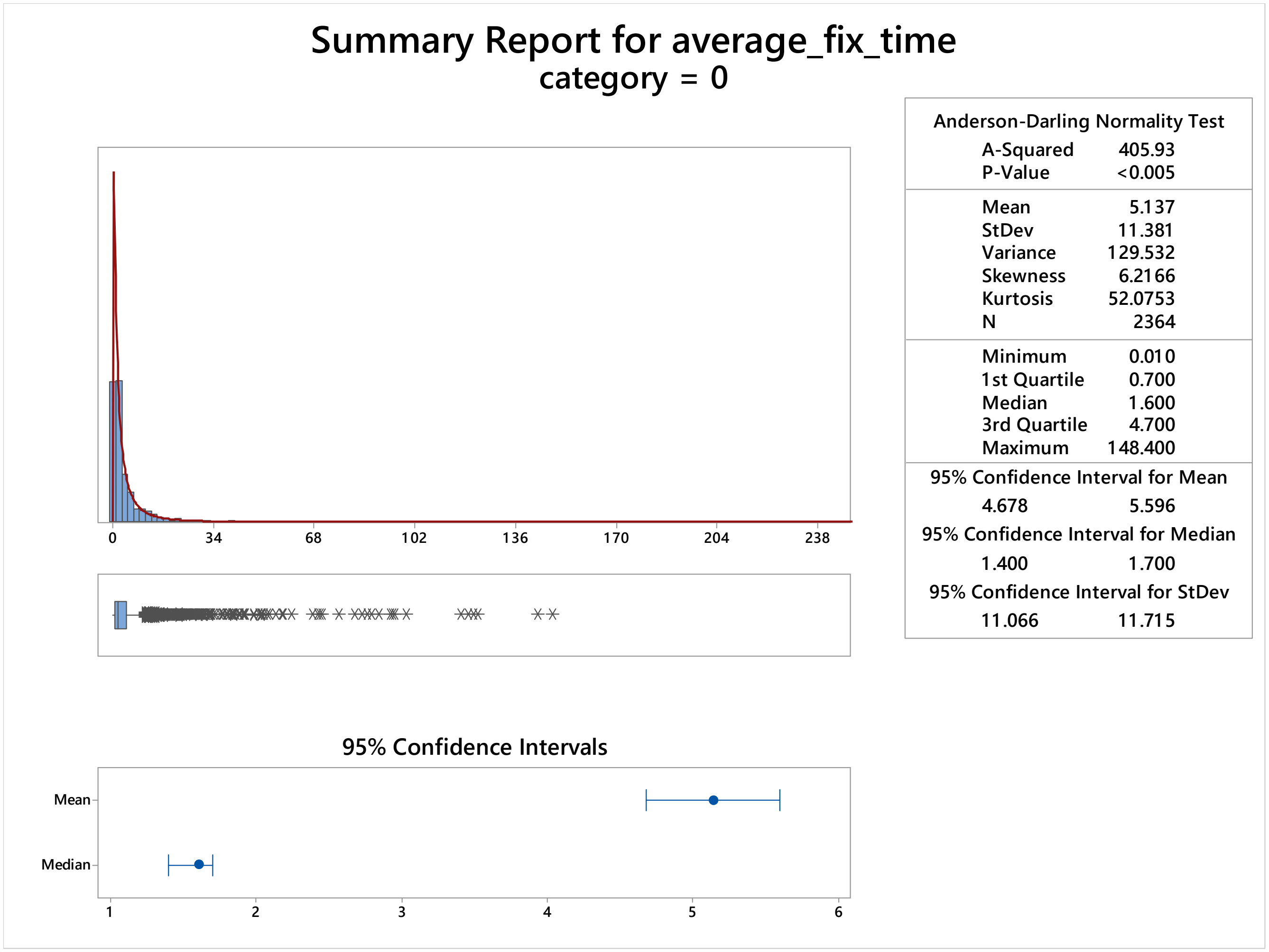
Anderson-Darling Normality Test	
A-Squared	1.98
P-Value	<0.005
Mean	8.9100
StDev	15.9526
Variance	254.4854
Skewness	2.17088
Kurtosis	4.13177
N	10
Minimum	0.3000
1st Quartile	1.1500
Median	1.8000
3rd Quartile	9.2250
Maximum	48.2000
95% Confidence Interval for Mean	
-2.5018	20.3218
95% Confidence Interval for Median	
1.1315	11.4135
95% Confidence Interval for StDev	
10.9728	29.1232

Figure 17: Tool_B_1 Defects, Average Fix Time, Compile



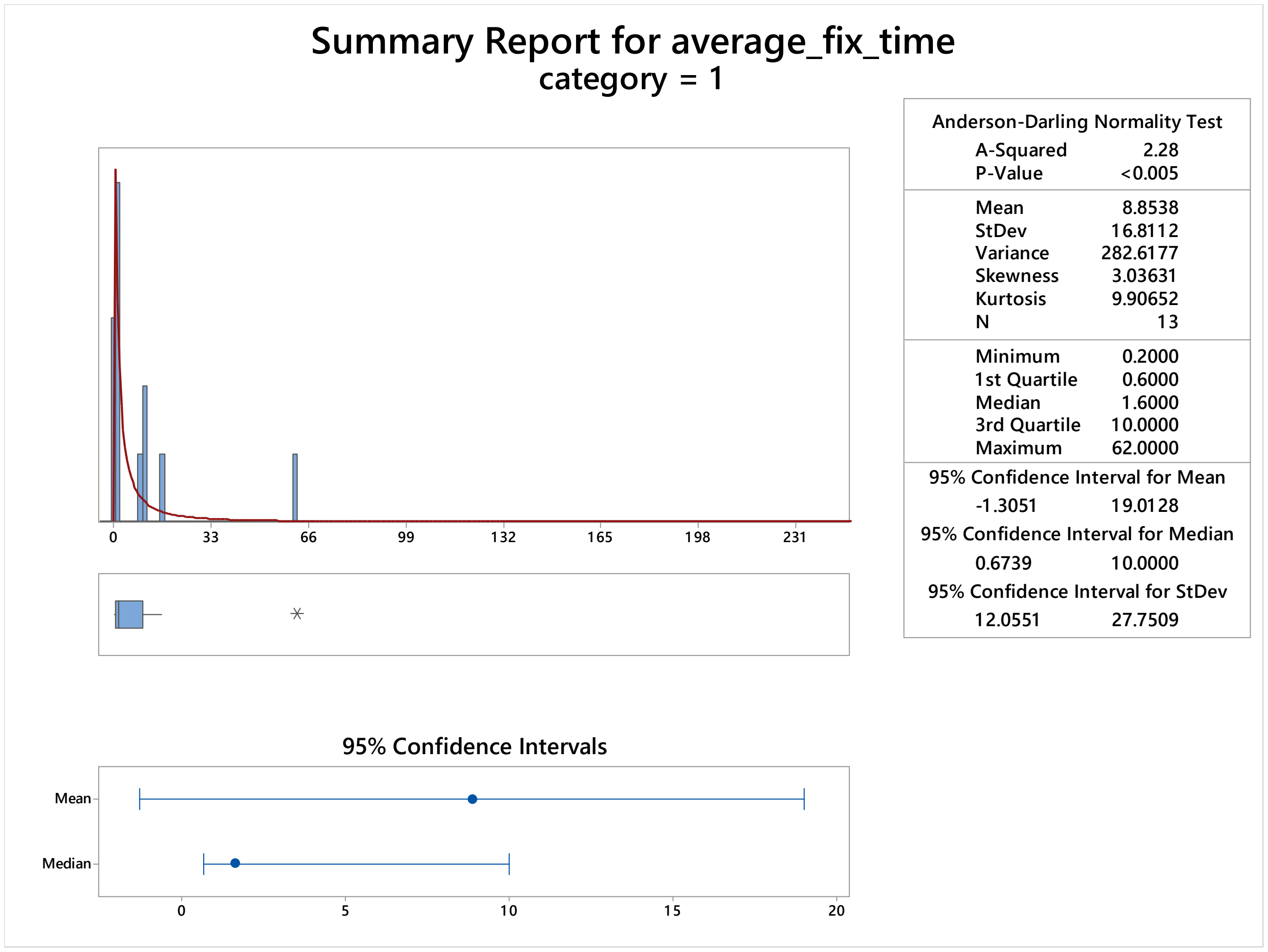
Anderson-Darling Normality Test	
A-Squared	0.54
P-Value	0.142
Mean	1.7118
StDev	1.2757
Variance	1.6274
Skewness	1.48448
Kurtosis	3.33182
N	17
Minimum	0.2000
1st Quartile	0.7500
Median	1.6000
3rd Quartile	2.3500
Maximum	5.4000
95% Confidence Interval for Mean	
1.0559	2.3677
95% Confidence Interval for Median	
0.9000	2.2928
95% Confidence Interval for StDev	
0.9501	1.9415

Figure 18: Tool_B_2 Defects, Average Find and Fix Time, Compile



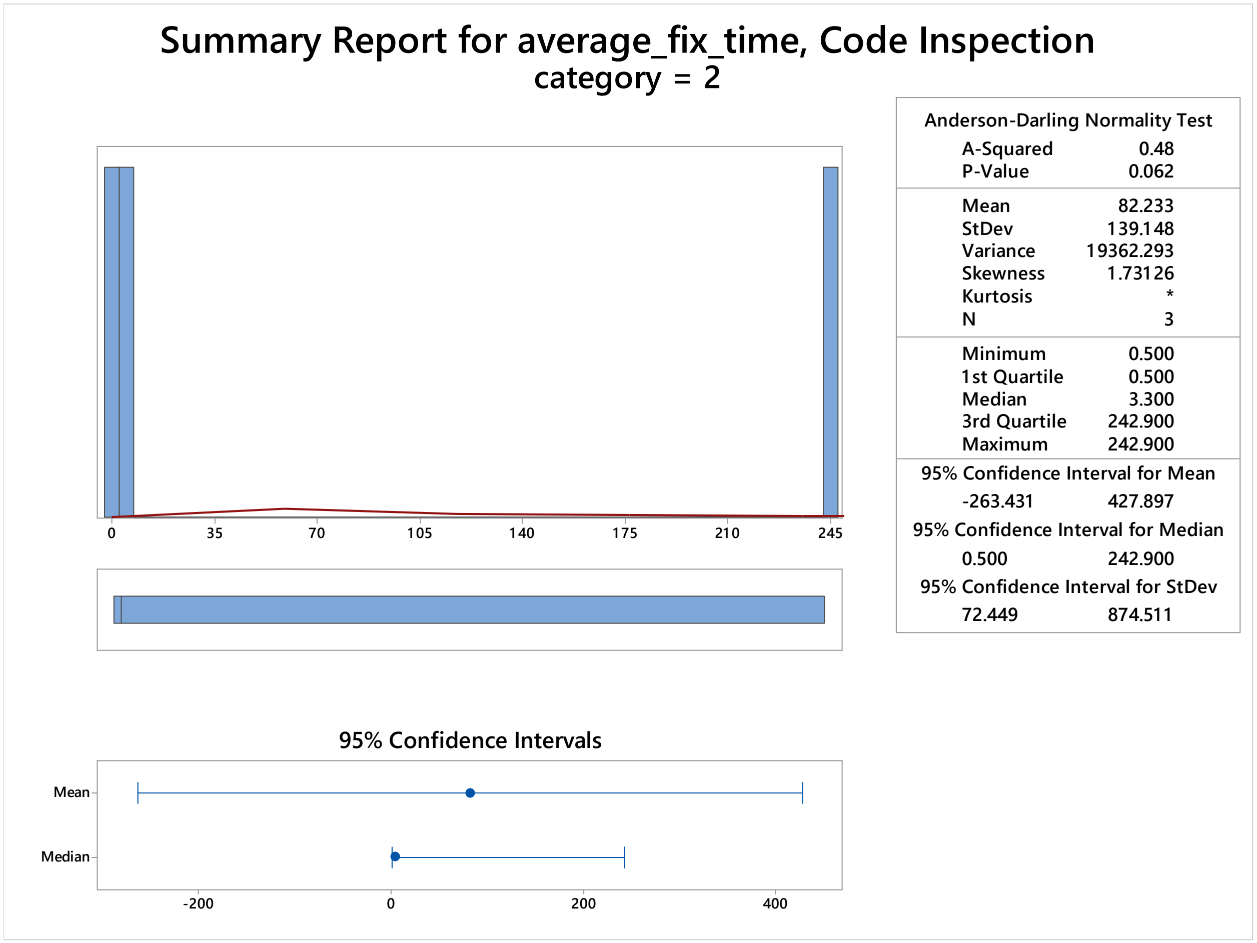
Anderson-Darling Normality Test	
A-Squared	405.93
P-Value	<0.005
Mean	5.137
StDev	11.381
Variance	129.532
Skewness	6.2166
Kurtosis	52.0753
N	2364
Minimum	0.010
1st Quartile	0.700
Median	1.600
3rd Quartile	4.700
Maximum	148.400
95% Confidence Interval for Mean	
4.678	5.596
95% Confidence Interval for Median	
1.400	1.700
95% Confidence Interval for StDev	
11.066	11.715

Figure 19: Other Defects, Fix Time Distribution, Code Inspect



Anderson-Darling Normality Test	
A-Squared	2.28
P-Value	<0.005
Mean	8.8538
StDev	16.8112
Variance	282.6177
Skewness	3.03631
Kurtosis	9.90652
N	13
Minimum	0.2000
1st Quartile	0.6000
Median	1.6000
3rd Quartile	10.0000
Maximum	62.0000
95% Confidence Interval for Mean	
-1.3051	19.0128
95% Confidence Interval for Median	
0.6739	10.0000
95% Confidence Interval for StDev	
12.0551	27.7509

Figure 20: Tool_B_1, Defect Fix Time Distribution, Code Inspect



Anderson-Darling Normality Test

A-Squared	0.48
P-Value	0.062
Mean	82.233
StDev	139.148
Variance	19362.293
Skewness	1.73126
Kurtosis	*
N	3
Minimum	0.500
1st Quartile	0.500
Median	3.300
3rd Quartile	242.900
Maximum	242.900

95% Confidence Interval for Mean

-263.431	427.897

95% Confidence Interval for Median

0.500	242.900

95% Confidence Interval for StDev

72.449	874.511

Figure 21: Tool_B_2 Defect Find and Fix Distribution, Code Inspect

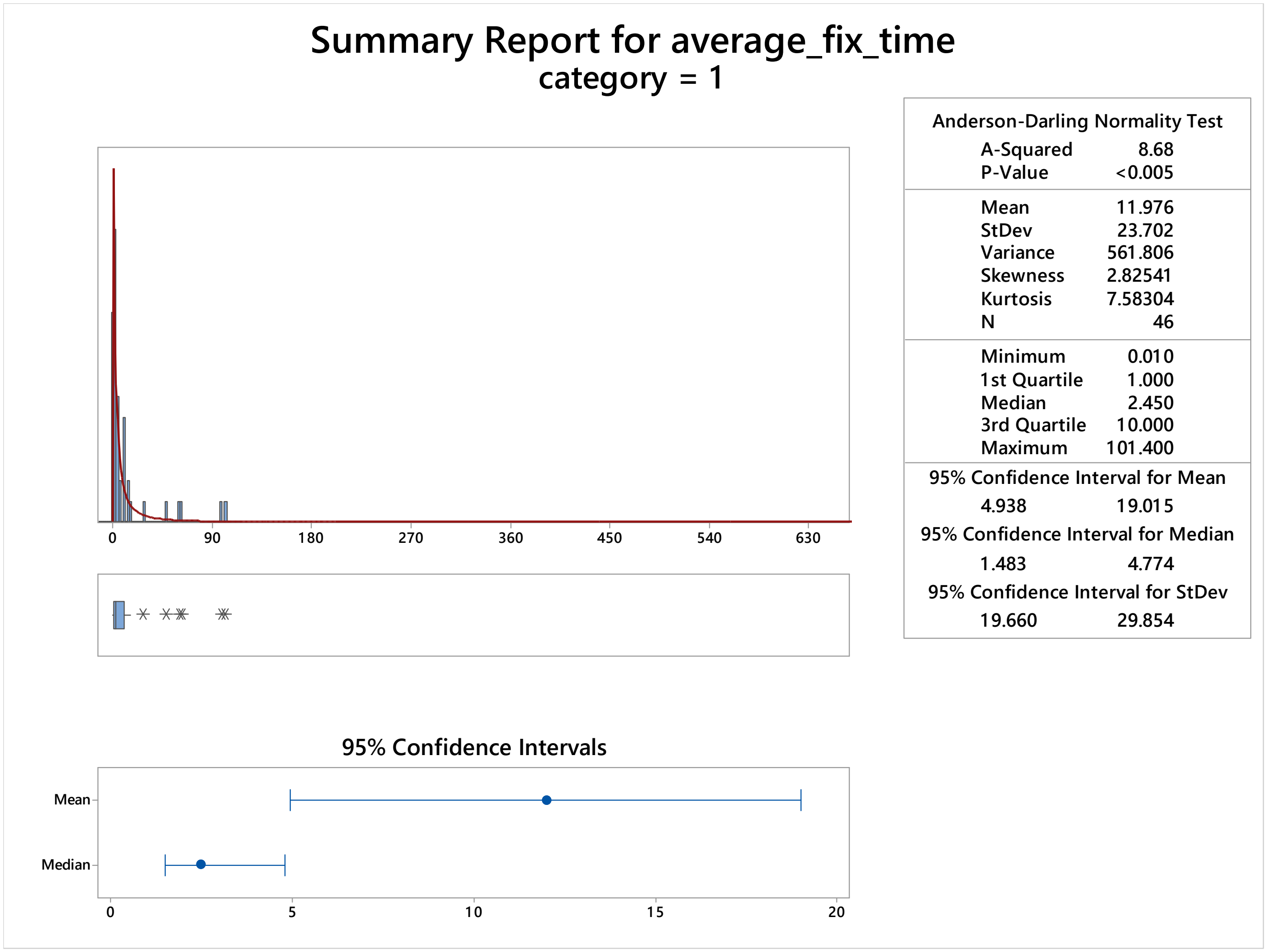


Figure 22: Tool_B_1 Defect Find and Fix Distribution

In this project we observed a strong correlation between effectiveness of the personal code review and the peer code inspections (see Figure 23). We normally expect to see this correlation (but do not always) because similar skills are applied. The range of review and inspection yields is very wide.

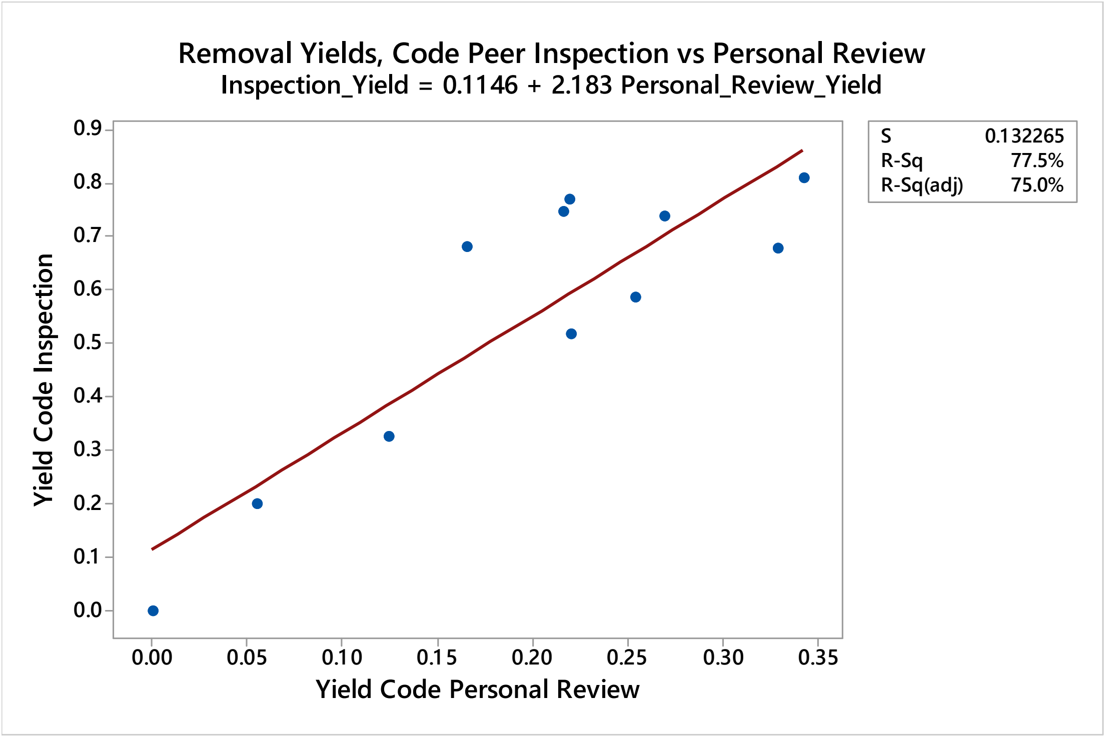


Figure 23: Inspection Phase Yield vs. Personal Review Phase Yield (Organization B)

Table 28: Defect Type Frequencies Found During Development Phase (Organization B)

Defect Type	After Development	Code Inspection	Code Review	Compile	Test	Row Totals
Assignment				1		1
Checking				2		2
DESIGN - Interface				1		1
DESIGN - Stand-ards			1	1		2
DEV - Assignment	1			2		3
DEV - Interface				1		1
DEV - Standards		2		3		5
DEV - Syntax				1		1
Function				1		1
Interface				4	1	5
Syntax		2			1	3
Column Totals	1	4	1	17	2	25

Figure 24, a scatterplot of project average code review yields versus code review rates in lines of code per hour, shows very weak correlation between the yield in code review and the rate at which code was reviewed. Nonetheless, though the review rate never exceeded 500 LOC per hour, about half the reviews exceeded the 200 LOC per hour recommendation. This lack of correlation is observed at a project level; the individual developer or component levels were not examined. Although a plausible explanation is that there was large variance among individual developers in review effectiveness, this analysis was not pursued because it was beyond the scope of our research questions.

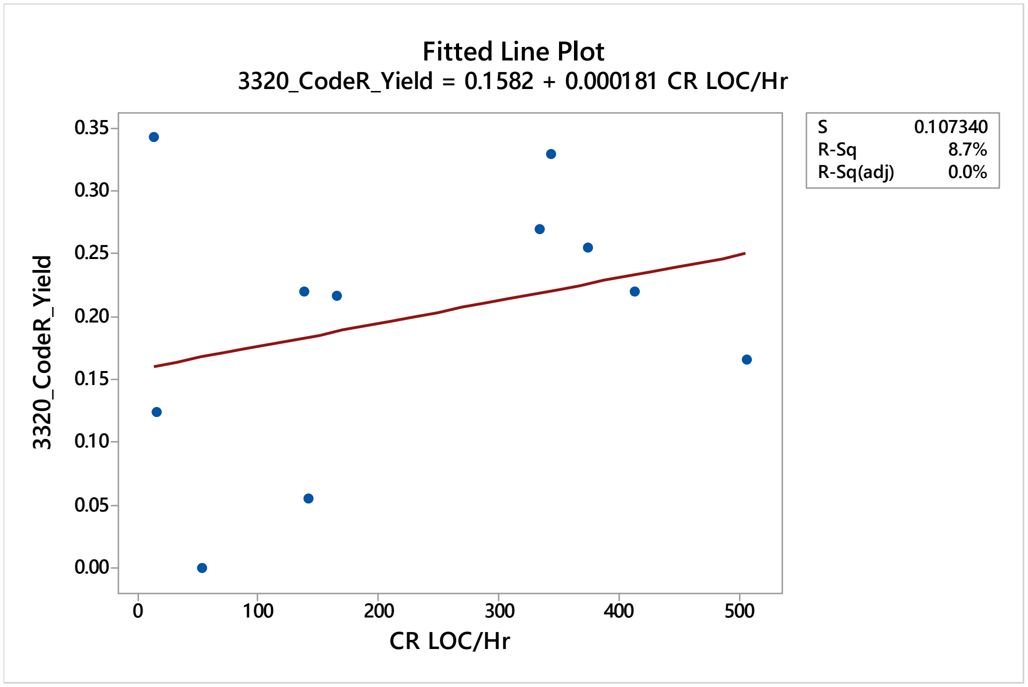


Figure 24: Personal Review Rate vs. Code Review Yield (Organization B)

We had identified a set of projects before the tools were inserted into the development with the intention of analyzing the parameters pre- and post-. However, the pre-post differences were obscured by the large overall performance variation. We therefore adopted a different analysis strategy to estimate tool effectiveness. Our key observation from the "before" projects was that no defects were logged to the "compile" activity. After verifying with project members, we attributed all compile activity to the tools in the "post" projects. That is, "compile" was used only as a tool activity, not to record actual compile time. We then used the data that was explicitly activity as "static analysis" to estimate adjustments to the other code review and code inspection phases. The statistical parameters are included in Table 29.

Table 29: Performance with Tool_B_2 and Tool_B_1 Static Analysis (Organization B)

Phase	No.Defect. Phase.Rate [LOC/Hr]	No.Defect. Phase.Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
0000_BeforeDev		0.0000		0.0000	
1000_Misc	128.0464	0.0078	0.0000		
1100_Strat	903.5513	0.0011	0.0000		
1150_Planing	164.2225	0.0061	0.0345		0.4517
3000_Req	209.0475	0.0048	1.3872		0.4517
3020_ReqR	612.7987	0.0016	0.0112	0.2066	0.0370
3040_ReqI	276.1358	0.0033	0.0681	0.7856	0.0582
3100_HLD	2238.8472	0.0004	1.4939		0.3050
3110_ITP	257.4165	0.0039	0.1553	0.0806	0.0398
3120_HLDR	13878.0127	0.0000	0.0000	0.0979	0.1513

Phase	No.Defect. Phase.Rate [LOC/Hr]	No.Defect. Phase.Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
3140_HLDI	1995.2280	0.0004	0.0182	0.2000	0.1627
3200_DLD	108.8589	0.0092	3.4666		0.2192
3210_TD	288.3239	0.0034	1.4073		0.1842
3220_DLDR	327.9854	0.0020	0.0869	0.2583	0.1002
3220_DLDI	86.2034	0.0098	0.1079	0.4818	0.1194
3300_Code	95.8922	0.0099	3.3868		0.1574
3320_CodeR	332.3528	0.0019	0.0972	0.2528	0.0914
3330_Compile	1713.4691	0.0004	0.1723	0.1043	0.0551
3340_CodeI	64.4342	0.0133	0.0801	0.6539	0.1007
3350_UTest	160.2417	0.0043	0.0557	0.5511	0.3120
3400_Test-CaseDevel	0.0000	0.0000	0.0000	0.0000	0.0000
4010_BITest	122.4605	0.0080	0.0515	0.2166	0.1409
4015_xxxx	0.0000	0.0000	0.0000	0.0000	0.0000
4030_STest	343.4037	0.0024	0.0000	0.4000	0.1476
4040_Doc	608.1998	0.0016	0.2001	0.1250	0.2403
4050_ATest	95.3808	0.0105	0.0000	0.4000	0.0137
5000_PM	1296.2678	0.0008	0.0000		
6100_PLife	12385.4713	0.0000	0.0000	0.4000	4.3500

Table 30: Performance without Tool_B_2 or Tool_B_1 Static Analysis (Organization B)

Phase	No.Defect. Phase.Rate {LOC/Hr]	No.Defect. Phase.Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
0000_BeforeDev		0.0000		0.0000	
1000_Misc	128.0464	0.0078	0.0000		
1100_Strat	903.5513	0.0011	0.0000		
1150_Planing	164.2225	0.0061	0.0345		0.4517
3000_Req	209.0475	0.0048	1.3872		0.4517
3020_ReqR	612.7987	0.0016	0.0112	0.2066	0.0370
3040_ReqI	276.1358	0.0033	0.0681	0.7856	0.0582
3100_HLD	2238.8472	0.0004	1.4939		0.3050
3110_ITP	257.4165	0.0039	0.1553	0.0806	0.0398
3120_HLDR	13878.0127	0.0000	0.0000	0.0979	0.1513
3140_HLDI	1995.2280	0.0004	0.0182	0.2000	0.1627
3200_DLD	108.8589	0.0092	3.4666		0.2192
3210_TD	288.3239	0.0034	1.4073		0.1842

Phase	No.Defect. Phase.Rate {LOC/Hr]	No.Defect. Phase.Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/Defect]
3220_DLDR	327.9854	0.0020	0.0869	0.2583	0.1002
3220_DLDI	86.2034	0.0098	0.1079	0.4818	0.1194
3300_Code	95.8922	0.0099	3.3868		0.1574
3320_CodeR	332.3528	0.0019	0.0972	0.2400	0.0914
3330_Compile	0.0000	0.0000	0.0000	0.0000	0.0000
3340_CodeI	64.4342	0.0133	0.0801	0.6300	0.1007
3350_UTest	160.2417	0.0043	0.0557	0.5511	0.3120
3551xxxx	0.0000	0.0000	0.0000	0.0000	0.0000
3400_Test-CaseDevel		0.0000			
4010_BITest	122.4605	0.0080	0.0515	0.2166	0.1409
4015_xxxx	0.0000	0.0000	0.0000	0.0000	0.0000
4030_STest	343.4037	0.0024	0.0000	0.4000	0.1476
4040_Doc	608.1998	0.0016	0.2001	0.0000	0.2403
4050_ATest	95.3808	0.0105	0.0000	0.4000	0.0137
5000_PM	1296.2678	0.0008	0.0000		
6100_PLife	12385.4713	0.0000	0.0000	0.4000	4.3500

Table 31: Number of Defects Removed per Phase with Static Analysis (Organization B)

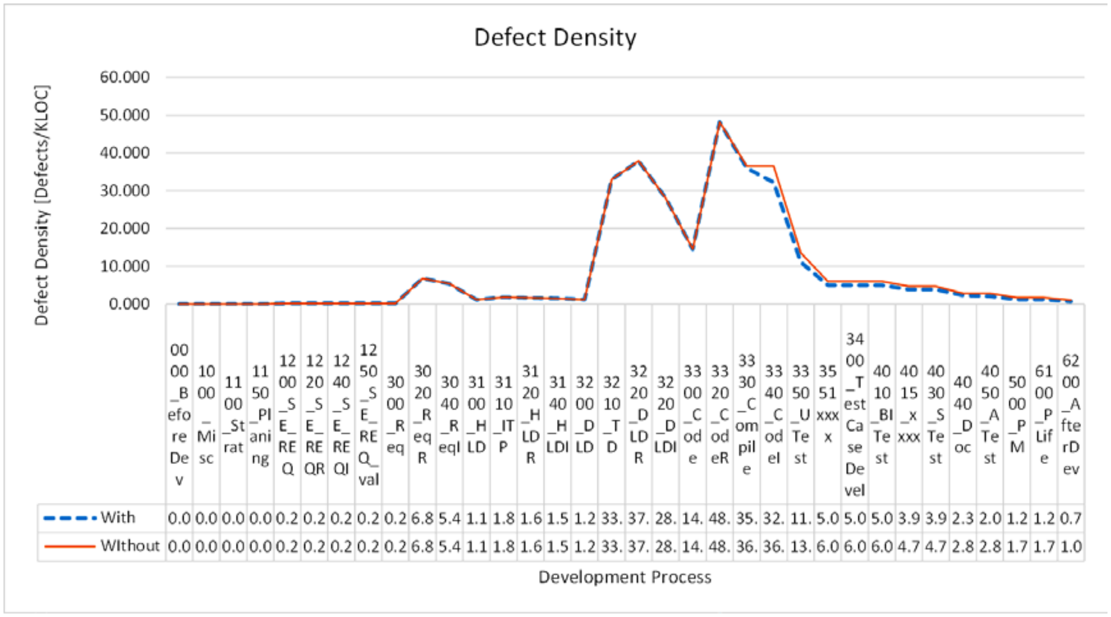
Injected Phase	Design Review	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	1070	1607	271	85	19	220	246	2	20			7	3547
Design Review	17	6	2	1		1	1						28
Design Inspect		48	45	13	1	20	7		1				135
Code			6	1204	361	2160	230	11	27		1	2	4002
Code Review				8	14	4	6						32
Code Inspect						84	65	1	1				151
Compile						10	1						11
Test Devel			8	6		4	82	128	298				526
Test							19	4	15			1	39
Int Test								7	34	5			46
Total	1087	1661	332	1317	395	2503	657	153	396	5	1	10	8517

Table 32: Effort Spent on Defect Removal with Static Analysis (Organization B)

Injected Phase	Design Review	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	6172.8	10798.3	2378	1008.9	150.9	1981.2	6987	30.9	1121			830.2	31458.7
Design Review	198.7	23.1	2.9	21.8		2.5	23.7						272.7
Design Inspect		209.6	141.6	124.2	1.2	129	30.5		14.4				650.5
Code			23.3	5169.4	1091.1	10868.7	2508	586.9	984.3		99.8	56.8	21388.1
Code Review				33.4	23.9	34.6	30.7						122.6
Code Inspect						455.5	604.9	182	11				1253.4
Compile						36.6	11						47.6
Test Devel			51.8	42.3		129.8	127.4	347.8	752.7				1451.8
Test							212.8	1.7	82.6			3.3	300.4
Int Test								9.4	151.4	4.1			164.9
Total	6371.5	11031	2598	6400	1267.1	13637.9	10535	1158.7	3117	4.1	99.8	890.3	57110.7

Table 33: Average Amount of Effort to Find and Fix Defects without Static Analysis (Organization B)

Injected Phase	Design Review	Design Inspect	Code	Code Review	Compile	Code Inspect	Test	Int Test	Sys Test	Accept Test	Product Life	After Development	Total
Design	5.768971963	6.71954	8.776	11.869	7.9421	9.00545	28.4	15.45	56.04			118.6	31458.7
Design Review	11.69	3.85	1.45	21.8		2.5	23.7						272.7
Design Inspect		4.37	3.15	9.55	1.2	6.45	4.357		14.4				650.5
Code			3.88	4.29	3.02	5.03	10.9	53.35	36.46		99.8	28.4	21388.1
Code Review				4.18	1.70	8.65	5.1						122.6
Code In-spect						5.42	9.3	182	11				1253.4
Compile						3.66	11						47.6
Test Devel			6.475	7.05		32.45	1.55	2.75	2.53				1451.8
Test							11.2	0.425	5.51			3.3	300.4
Int Test								1.342	4.45	0.82			164.9
Total	5.86	6.64118	7.825	4.86	3.21	5.45	16.04	7.57	7.871	0.82	99.8	89.03	57110.7



	0000_BeforeDev	1000_Misc	1100_Strat	1150_Planing	1200_SE_REQ	1220_SE_REQR	1240_SE_REQI	1250_SE_REQ_val	3000_Req	3020_ReqR	3040_ReqI	3100_HLD	3110_ITP	3120_HLDR	3140_HLDI	3200_DLD	3210_TD
With	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	6.8	5.4	1.1	1.8	1.6	1.5	1.2	33.
WIthout	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.2	0.2	6.8	5.4	1.1	1.8	1.6	1.5	1.2	33.

	3220_DLDR	3220_DLDI	3300_Code	3320_CodeR	3330_Compile	3340_Codel	3350_UTest	3551xxxx	3400_TestCaseDevel	4010_BITest	4015_xxxx	4030_STest	4040_Doc	4050_ATest	5000_PM	6100_PLife	6200_AfterDev
With	37.	28.	14.	48.	35.	32.	11.	5.0	5.0	5.0	3.9	3.9	2.3	2.0	1.2	1.2	0.7
WIthout	37.	28.	14.	48.	36.	36.	13.	6.0	6.0	6.0	4.7	4.7	2.8	2.8	1.7	1.7	1.0

Figure 25: Defect Density per Phase with and without Static Analysis (Organization B)

Our analysis of the measured results by phase and the expected results by phase if the static analysis tools had not been used are shown in Figure 25 through Figure 29. For these projects in this organization, the effects were very small. Cumulative effort was slightly lower using the tools (see Figure 26) because effort increased by a tiny amount in the removal phases (see Figure 27), but was more than compensated for by the lower effort in test. Test effort was reduced because the defect density was reduced by a small amount prior to test. For phased defect removal, see Figure 30.

The cumulative defect flows cannot be easily distinguished graphically (see Figure 28 and Figure 29). We believe the effects of using these tools were positive both for escaped defect density and total effort, but the effect was modest. The vast majority of defects were removed using conventional techniques of review and test. A potential benefit of the tools is targeted removal of specific known weaknesses that might escape review, inspection, and test.

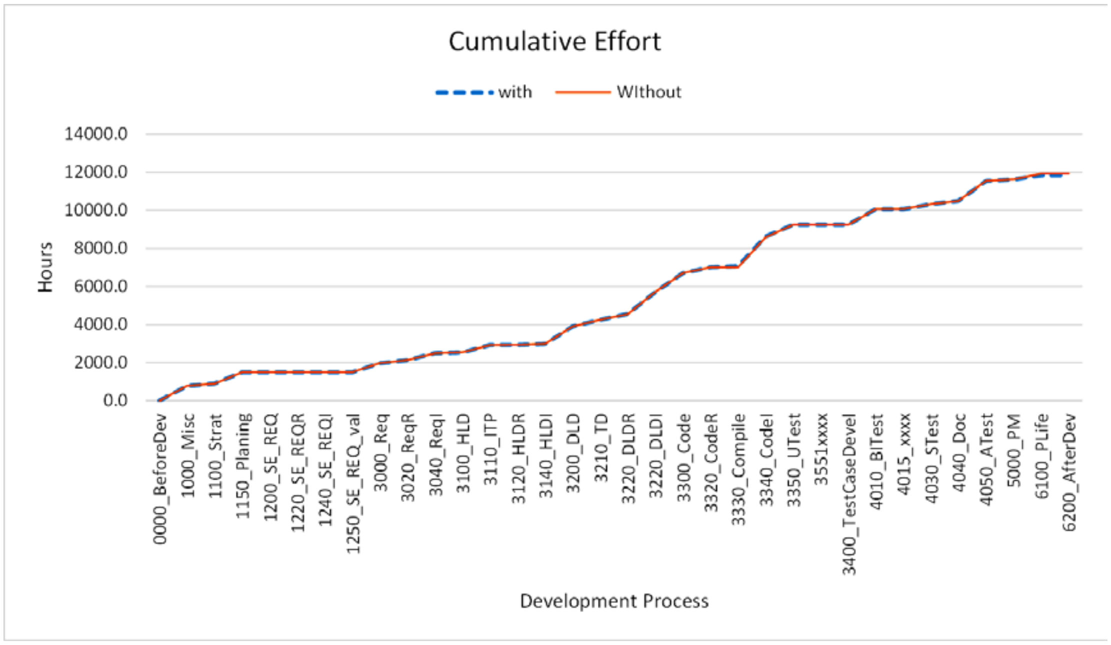


Figure 26: Cumulative Amount of Effort with and without Static Analysis (Organization B)

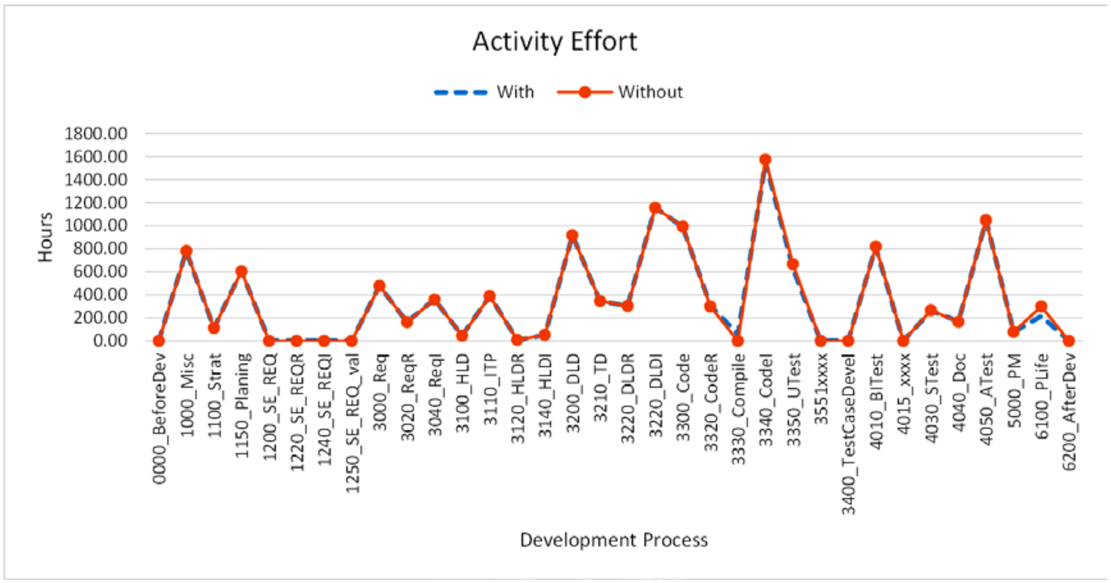


Figure 27: Team Effort by Phase with and without Static Analysis (Organization B)

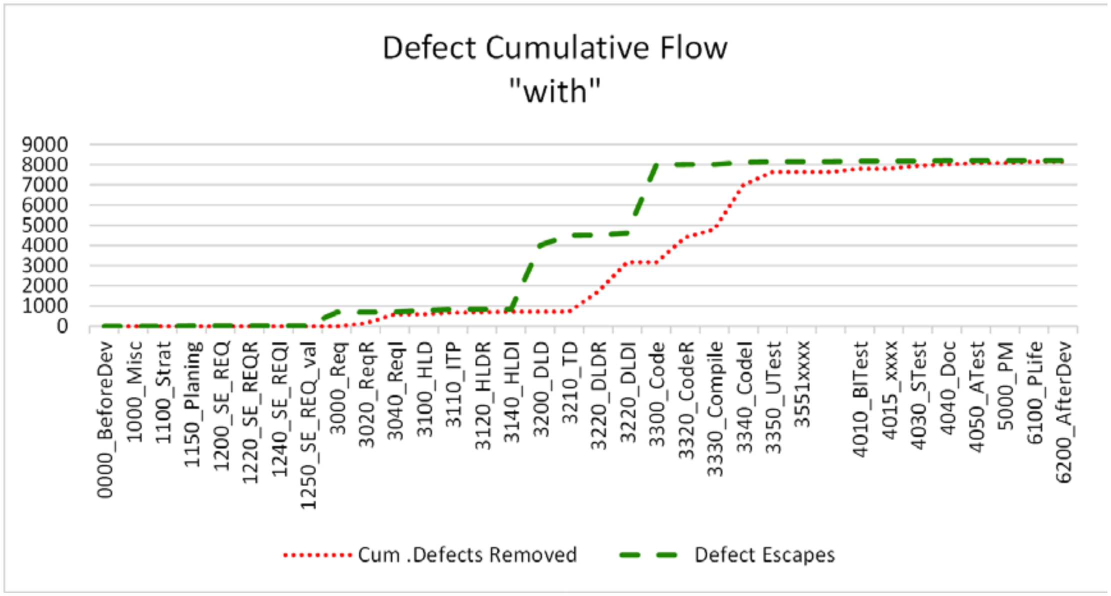


Figure 28: Cumulative Defect Flow with Static Analysis (Organization B)

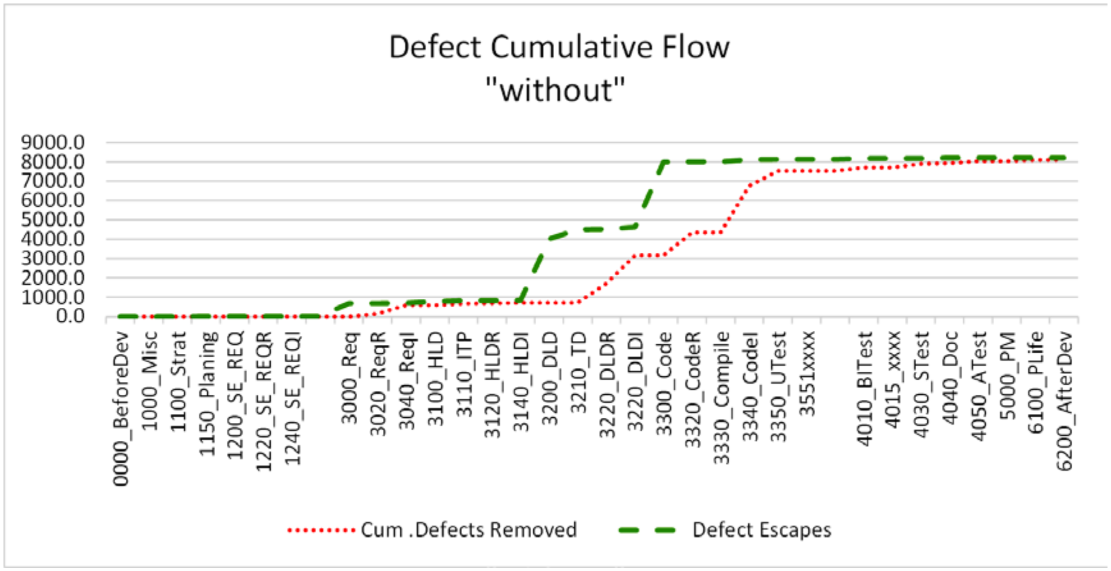


Figure 29: Cumulative Defect Flow without Static Analysis (Organization B)

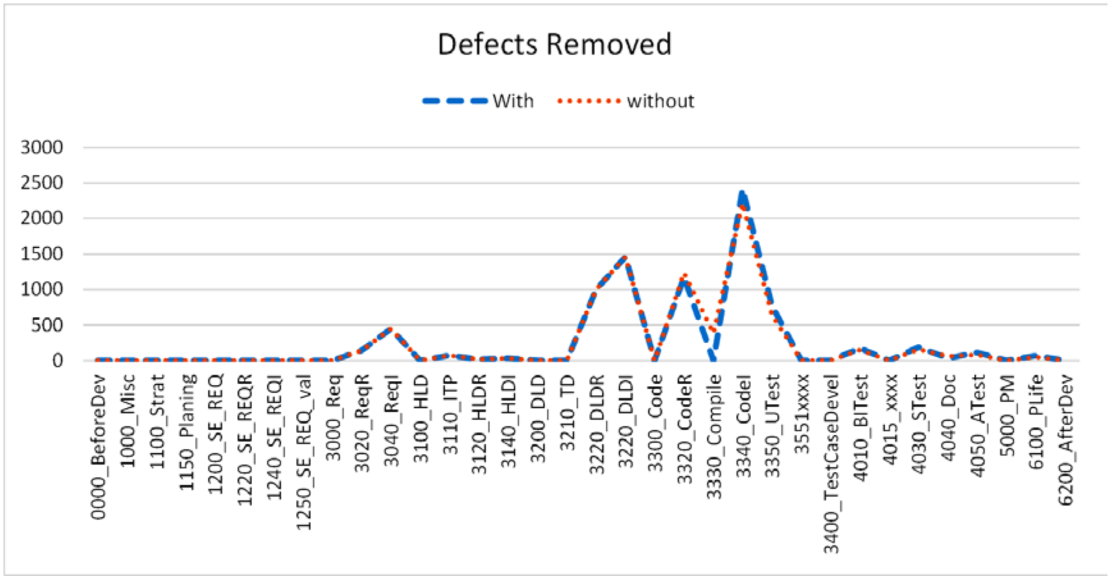


Figure 30: Defects Removed per Phase with and without Static Analysis during Personal Review Phase (Organization B)

4.3 Organization C

Organization C used a commercial tool that statically analyzed both the source code and the final binary. The tool was integrated into the build process and executed prior to test.

Related research questions for this case include the following:

- How much time does the project spend in each development process?
- What are the find and fix times for defects found by the various activities?

The phase effort question is partially addressed in Figure 31. The fraction of time in individual phases varied widely. This may have resulted from different processes, but can also result from differences in the specific work packages in the project. Figure 32 shows that all removal yields vary widely, but none as much as test. Descriptive statistics for yields are summarized in Table 34.

The find and fix time distribution for the static analysis tool is shown in Figure 34, with statistics provided in Table 35.

The majority of the defects were injected in code. The fix times for those defects by removal phase are shown in Figure 35. A small number of the defects were injected in prior projects. Following this is the distribution of the defect removal efforts by phase. The average values are used in the model. Again, we note that the distributions are highly skewed (approximately log-normal in frequency). A table summarizing the final parameters used in the model is included in Table 36.

Figure 31: Project Development Process Effort (Organization C)

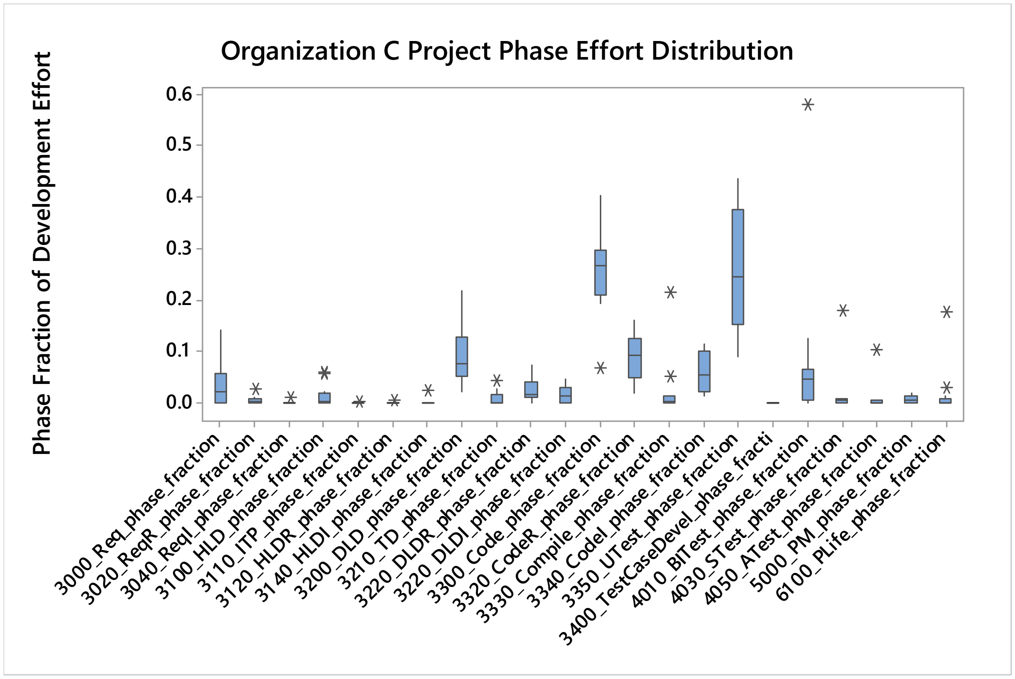


Figure 32: Project Defect Removal Yields (Organization C)

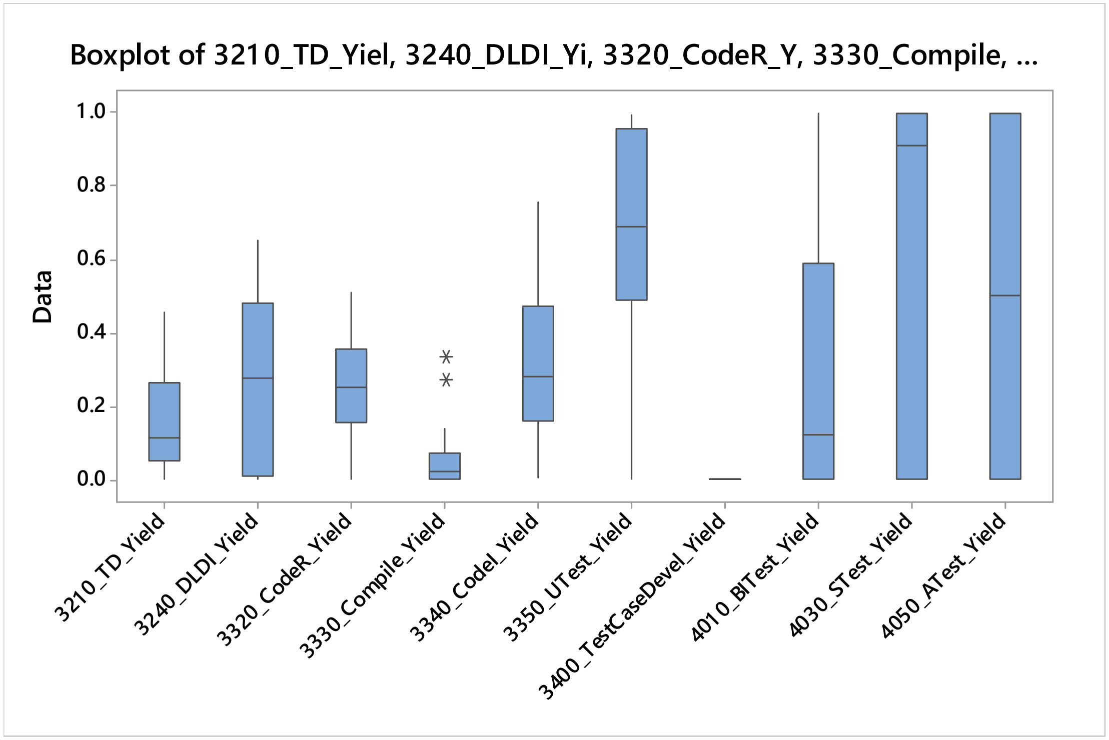
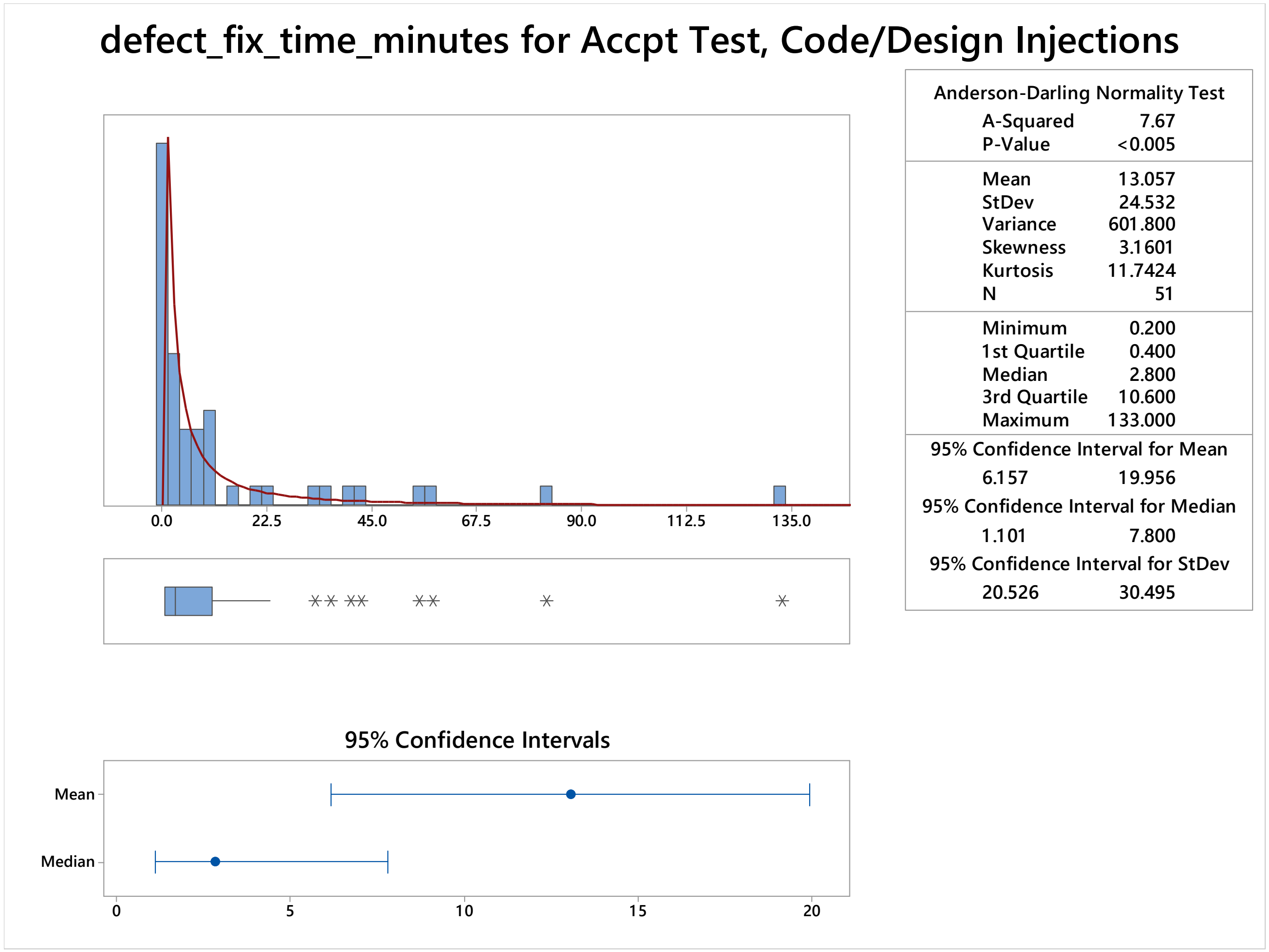


Table 34: Descriptive Statistics, Phase Yields (Organization C)

	N	N_missing	Mean	SE Mean	StDev	Minimum	Q1	Median	Q3	Maximum
3210_TD_Yield	14	0	0.16	0.04	0.13	0.00	0.05	0.12	0.27	0.46
3240_DLDI_Yield	14	0	0.27	0.06	0.24	0.00	0.01	0.28	0.48	0.65
3320_CodeR_Yield	14	0	0.25	0.04	0.14	0.00	0.15	0.25	0.35	0.51
3330_Compile_Yield	14	0	0.07	0.03	0.11	0.00	0.00	0.02	0.07	0.33
3340_CodeI_Yield	14	0	0.32	0.06	0.22	0.01	0.16	0.28	0.47	0.76
3350_UTest_Yield	14	0	0.65	0.08	0.31	0.00	0.49	0.69	0.96	0.99
3400_TestCaseDevel_Yield	14	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4010_BITest_Yield	14	0	0.32	0.10	0.37	0.00	0.00	0.12	0.59	1.00
4030_STest_Yield	13	1	0.53	0.14	0.50	0.00	0.00	0.91	1.00	1.00
4050_ATest_Yield	9	5	0.50	0.17	0.50	0.00	0.00	0.50	1.00	1.00

To address the question of how much effort is required to fix defects in each development activity, we collected the histograms, boxplots, and descriptive statistics of the find and fix times, shown in Figure 33 through Figure 45.

Figure 33 displays the find and fix time distribution for defects injected in code or design and removed in acceptance test. Figure 34 shows the distribution of defects explicitly coded as found in “static analysis” with the descriptive statistics included in Table 35. The acceptance test has a high uncertainty in the median value that cannot be distinguished from the static analysis. Nonetheless, the mean values (dominated by the more expensive events) indicate a statistically important difference in the mean values of find and fix time.



Anderson-Darling Normality Test	
A-Squared	7.67
P-Value	<0.005
Mean	13.057
StDev	24.532
Variance	601.800
Skewness	3.1601
Kurtosis	11.7424
N	51
Minimum	0.200
1st Quartile	0.400
Median	2.800
3rd Quartile	10.600
Maximum	133.000
95% Confidence Interval for Mean	
6.157	19.956
95% Confidence Interval for Median	
1.101	7.800
95% Confidence Interval for StDev	
20.526	30.495

Figure 33: Defect Find and Fix Time in Acceptance Test, for Code and Design Defects

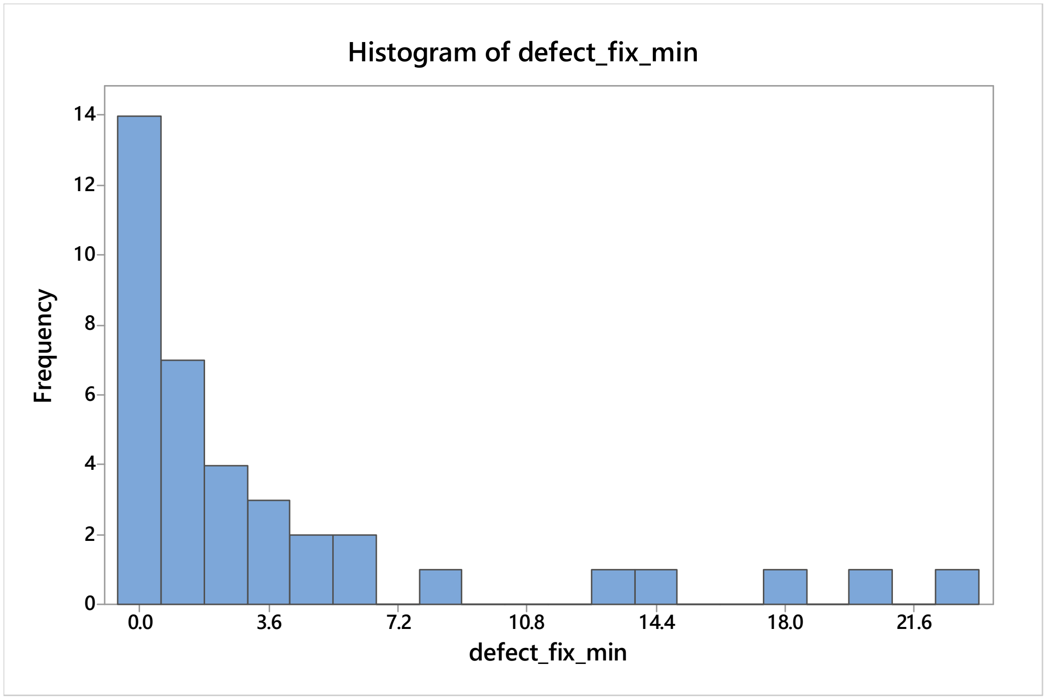


Figure 34: Static Analysis (Code and Binary) Defect Find/Fix Time

Table 35: Distribution of Static Analysis (Code and Binary) Defect Fix Times

Variable	N	N*	Mean	SE. Mean	StDev	Minimum	Q1	Median	Q3	Maximum
defect_fix_min	38	0	3.987	0.957	5.898	0.2	0.3	1.6	4.825	23.1

To model the scenarios with and without the static checker, we used data from the projects that explicitly attributed the defect finds and effort to acceptance test. Although the TSP uses this phase after system test, these teams used the acceptance test phase to collect defects between build and integration test and system test. We made the appropriate adjustment in the data.

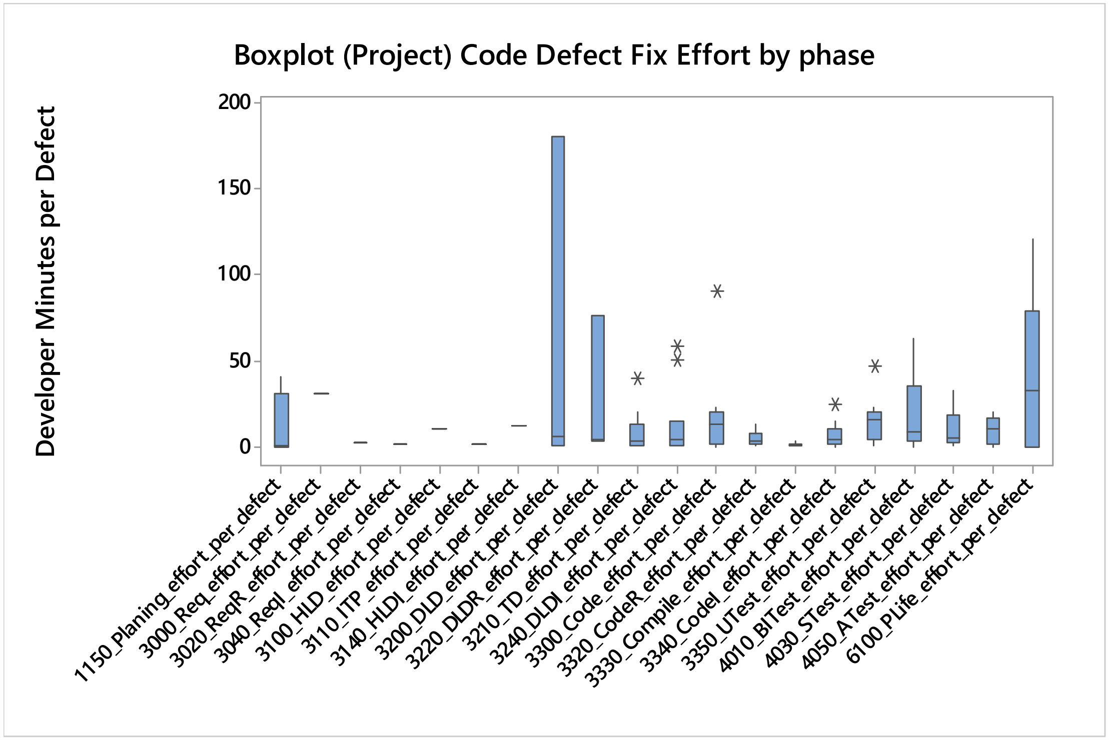


Figure 35: Project Distribution of Code Defect Find and Fix Time by Removal Phase (Organization C)

For defects explicitly marked as “static analysis” defects, we summarize the find and fix rates in Figure 34 and Table 35. These find and fix times can be compared to the distributions for all defects removed in each activity as shown in the boxplots in Figure 35 and the more detailed distributions that follow.

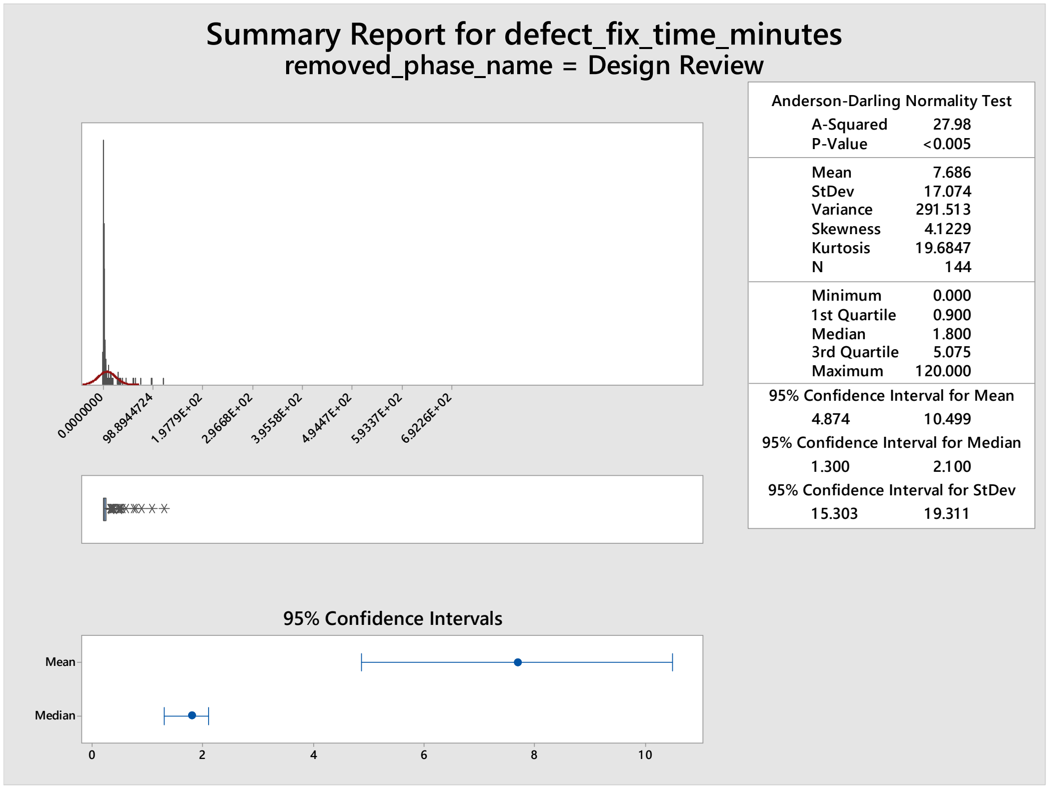


Figure 36: Defect Fix Time, Design Review (Organization C)

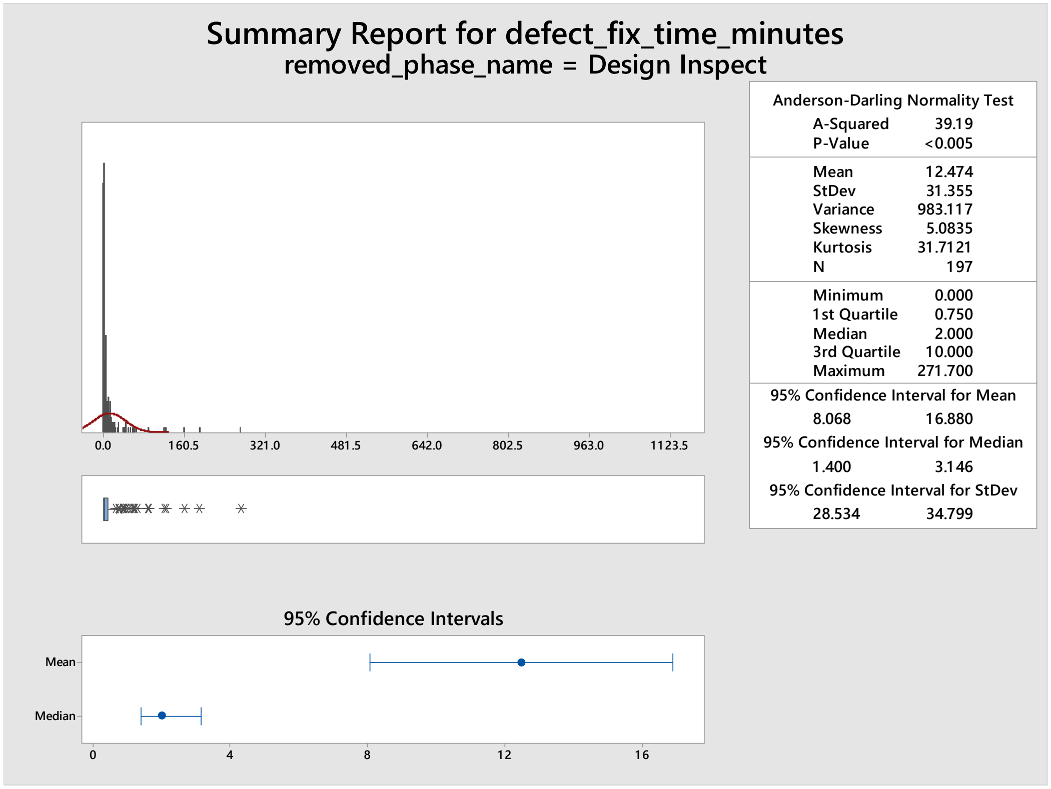
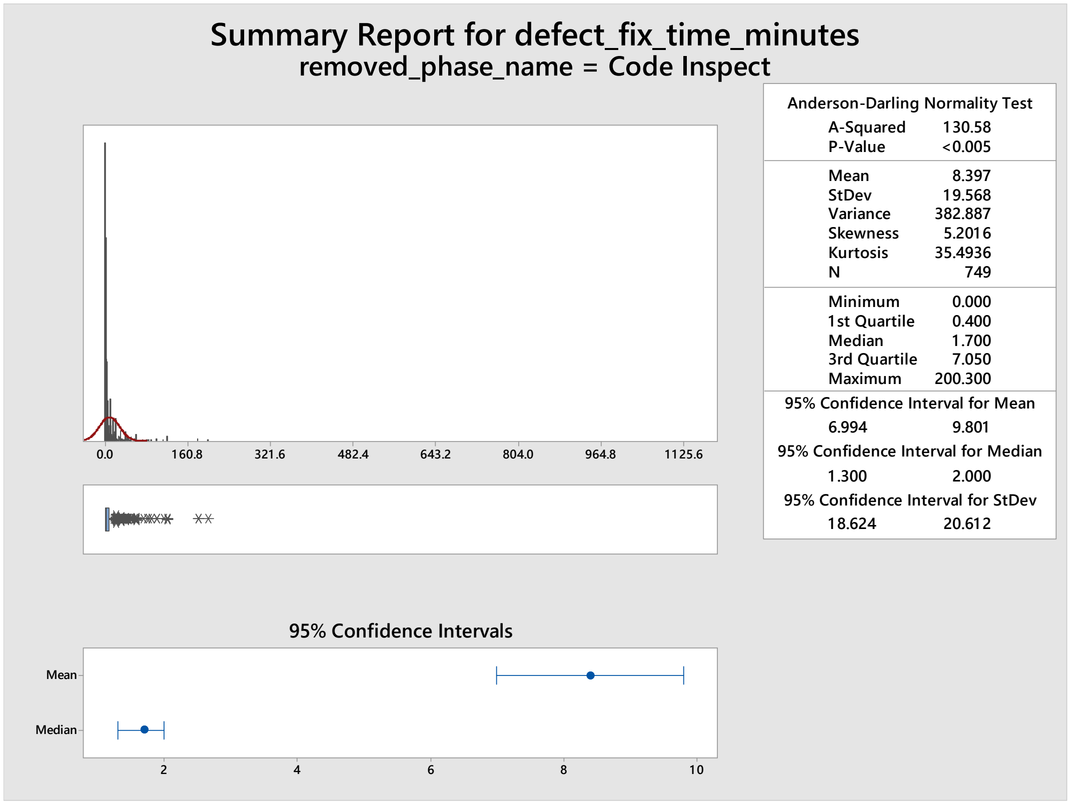


Figure 37: Defect Fix Time, Design Inspect (Organization C)

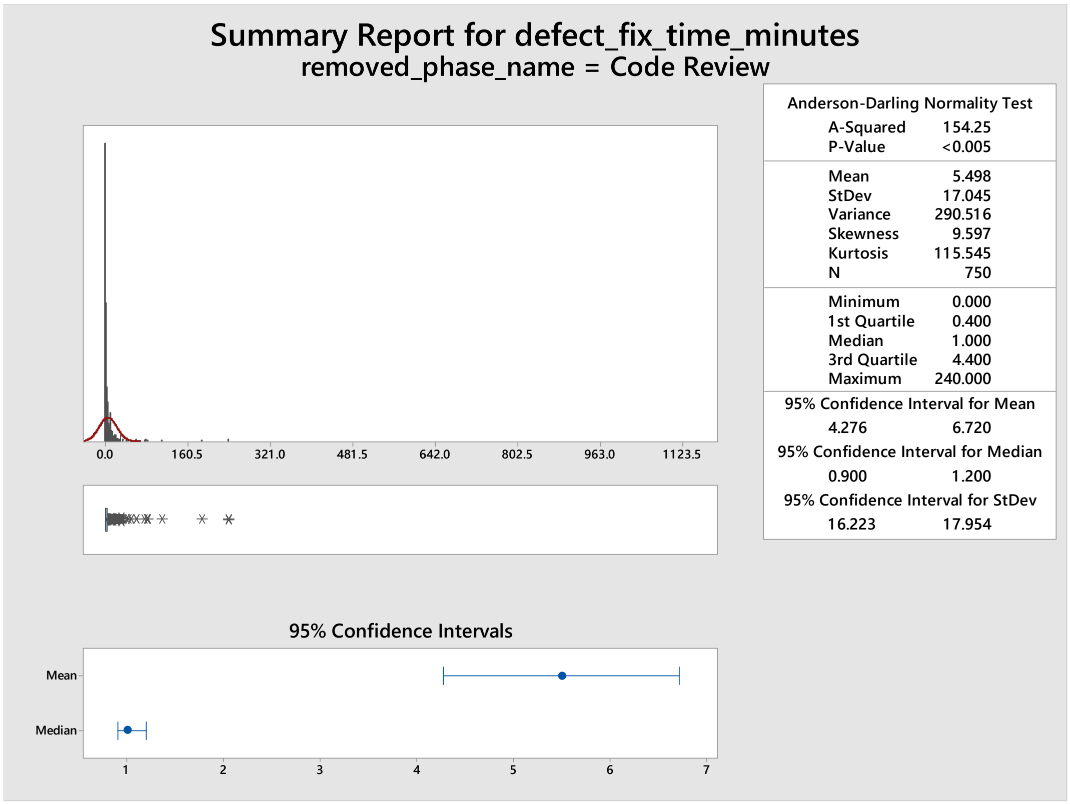


Anderson-Darling Normality Test

A-Squared	130.58
P-Value	<0.005
Mean	8.397
StDev	19.568
Variance	382.887
Skewness	5.2016
Kurtosis	35.4936
N	749
Minimum	0.000
1st Quartile	0.400
Median	1.700
3rd Quartile	7.050
Maximum	200.300

95% Confidence Interval for Mean
6.994 9.801
95% Confidence Interval for Median
1.300 2.000
95% Confidence Interval for StDev
18.624 20.612

Figure 38: Defect Fix Time, Code Inspect (Organization C)

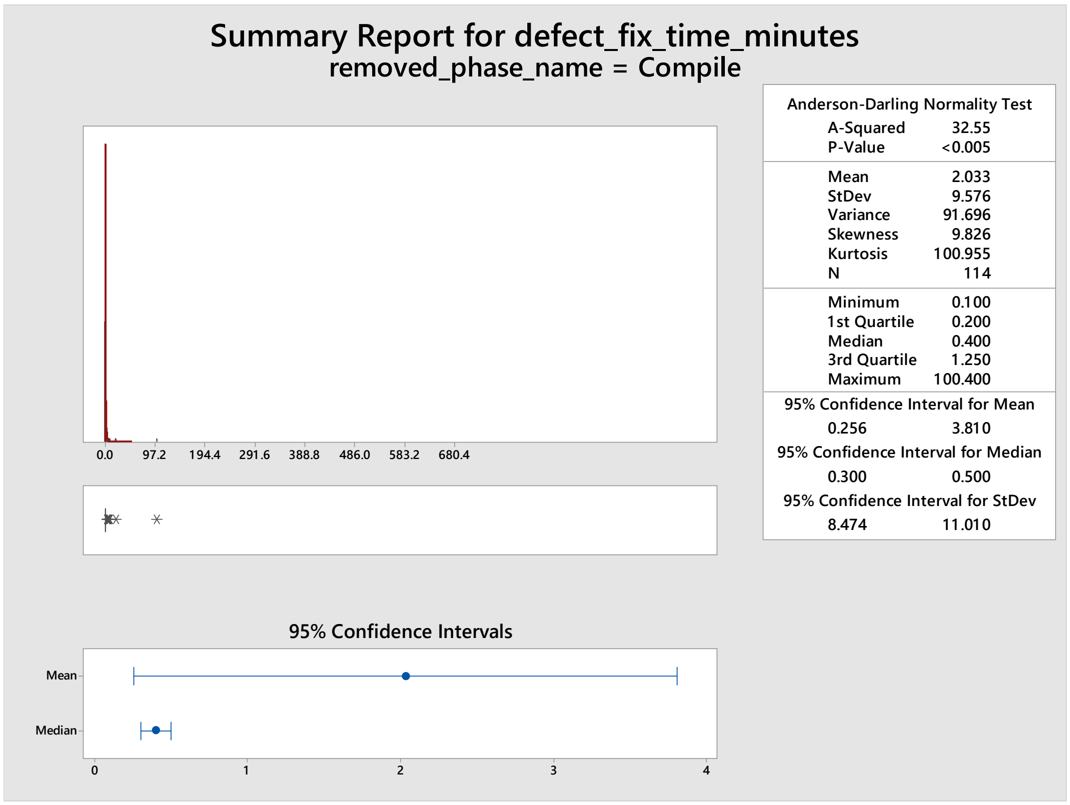


Anderson-Darling Normality Test

A-Squared	154.25
P-Value	<0.005
Mean	5.498
StDev	17.045
Variance	290.516
Skewness	9.597
Kurtosis	115.545
N	750
Minimum	0.000
1st Quartile	0.400
Median	1.000
3rd Quartile	4.400
Maximum	240.000

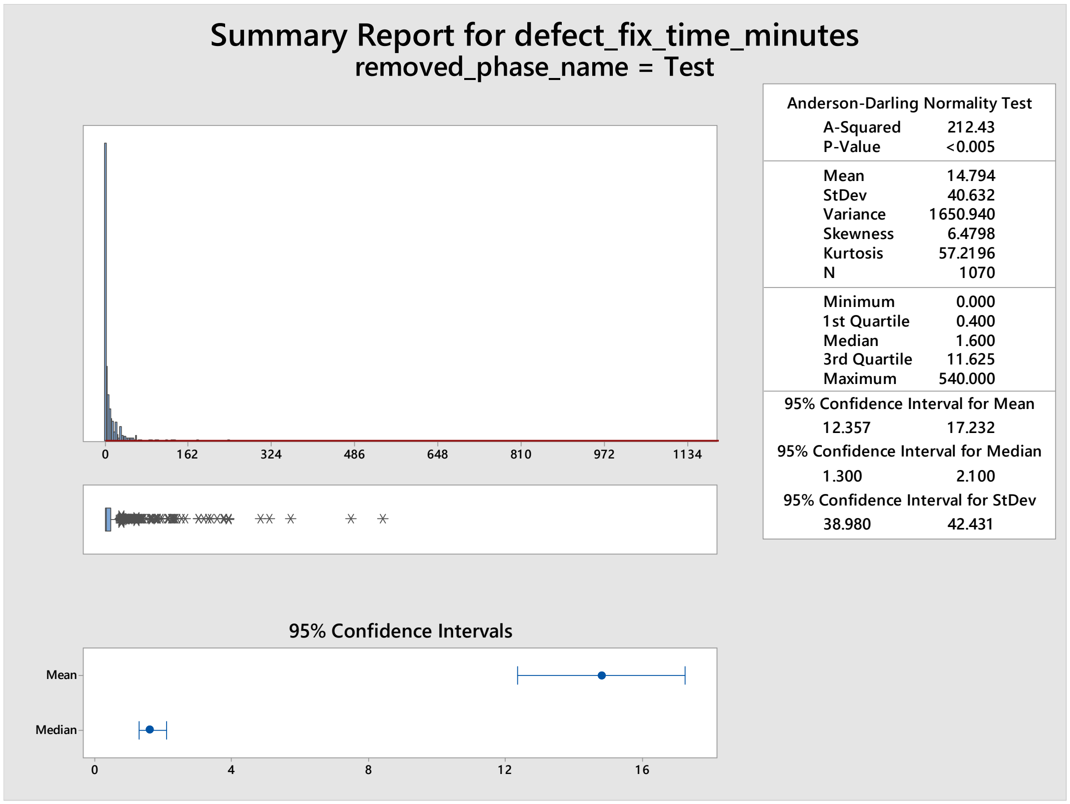
95% Confidence Interval for Mean
4.276 6.720
95% Confidence Interval for Median
0.900 1.200
95% Confidence Interval for StDev
16.223 17.954

Figure 39: Defect Fix Time, Code Review (Organization C)



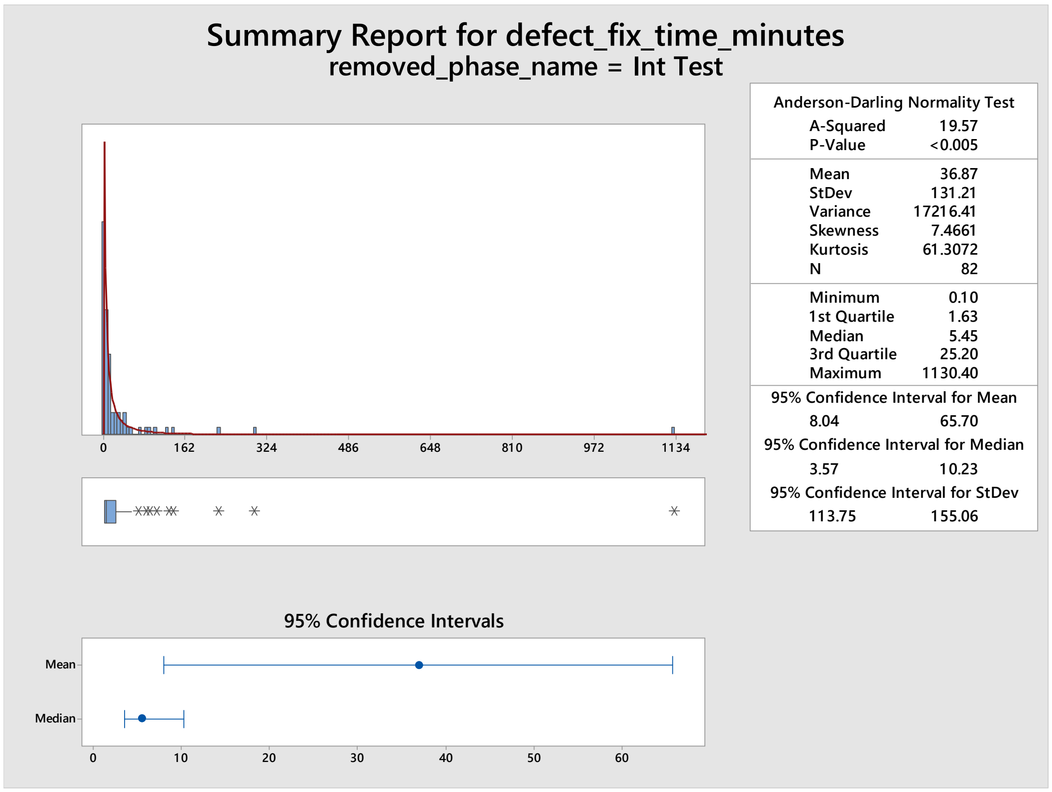
Anderson-Darling Normality Test	
A-Squared	32.55
P-Value	<0.005
Mean	2.033
StDev	9.576
Variance	91.696
Skewness	9.826
Kurtosis	100.955
N	114
Minimum	0.100
1st Quartile	0.200
Median	0.400
3rd Quartile	1.250
Maximum	100.400
95% Confidence Interval for Mean	
0.256	3.810
95% Confidence Interval for Median	
0.300	0.500
95% Confidence Interval for StDev	
8.474	11.010

Figure 40: Defect Fix Time, Compile (Organization C)



Anderson-Darling Normality Test	
A-Squared	212.43
P-Value	<0.005
Mean	14.794
StDev	40.632
Variance	1650.940
Skewness	6.4798
Kurtosis	57.2196
N	1070
Minimum	0.000
1st Quartile	0.400
Median	1.600
3rd Quartile	11.625
Maximum	540.000
95% Confidence Interval for Mean	
12.357	17.232
95% Confidence Interval for Median	
1.300	2.100
95% Confidence Interval for StDev	
38.980	42.431

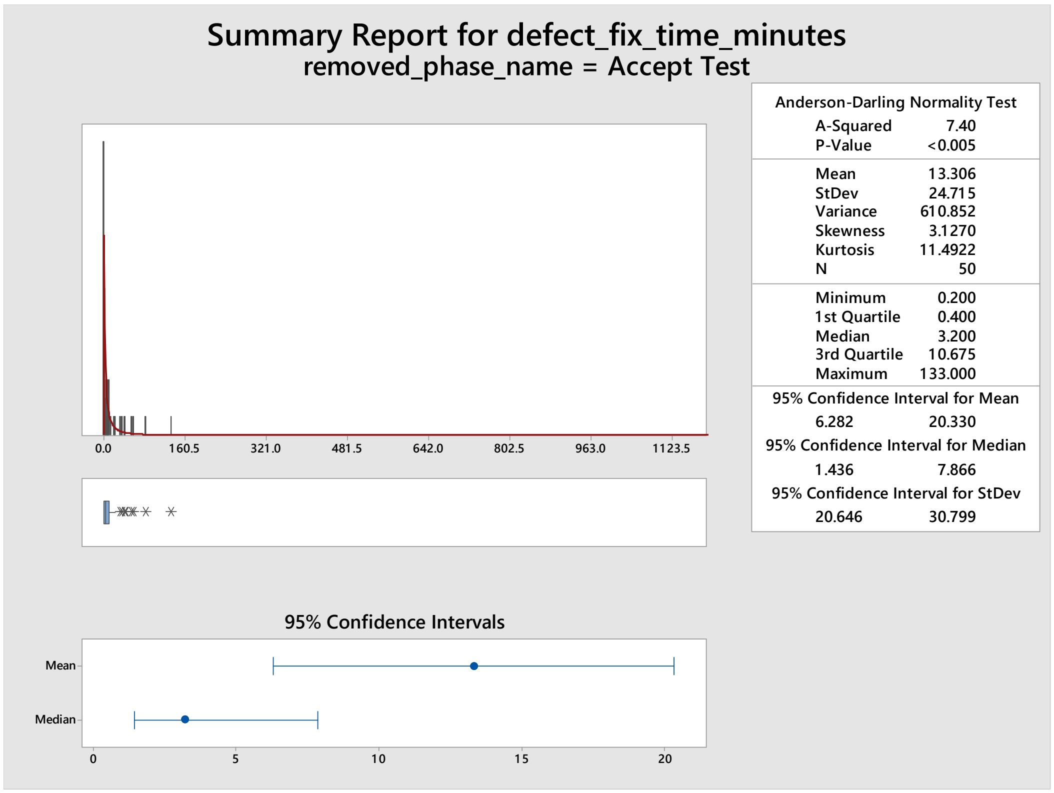
Figure 41: Defect Fix Time, Test (Organization C)



Summary Report for defect_fix_time_minutes
removed_phase_name = Int Test

Anderson-Darling Normality Test	
A-Squared	19.57
P-Value	<0.005
Mean	36.87
StDev	131.21
Variance	17216.41
Skewness	7.4661
Kurtosis	61.3072
N	82
Minimum	0.10
1st Quartile	1.63
Median	5.45
3rd Quartile	25.20
Maximum	1130.40
95% Confidence Interval for Mean	
8.04	65.70
95% Confidence Interval for Median	
3.57	10.23
95% Confidence Interval for StDev	
113.75	155.06

Figure 42: Defect Fix Time, Integration Test (Organization C)



Summary Report for defect_fix_time_minutes
removed_phase_name = Accept Test

Anderson-Darling Normality Test	
A-Squared	7.40
P-Value	<0.005
Mean	13.306
StDev	24.715
Variance	610.852
Skewness	3.1270
Kurtosis	11.4922
N	50
Minimum	0.200
1st Quartile	0.400
Median	3.200
3rd Quartile	10.675
Maximum	133.000
95% Confidence Interval for Mean	
6.282	20.330
95% Confidence Interval for Median	
1.436	7.866
95% Confidence Interval for StDev	
20.646	30.799

Figure 43: Defect Fix Time, Acceptance Test (Organization C)

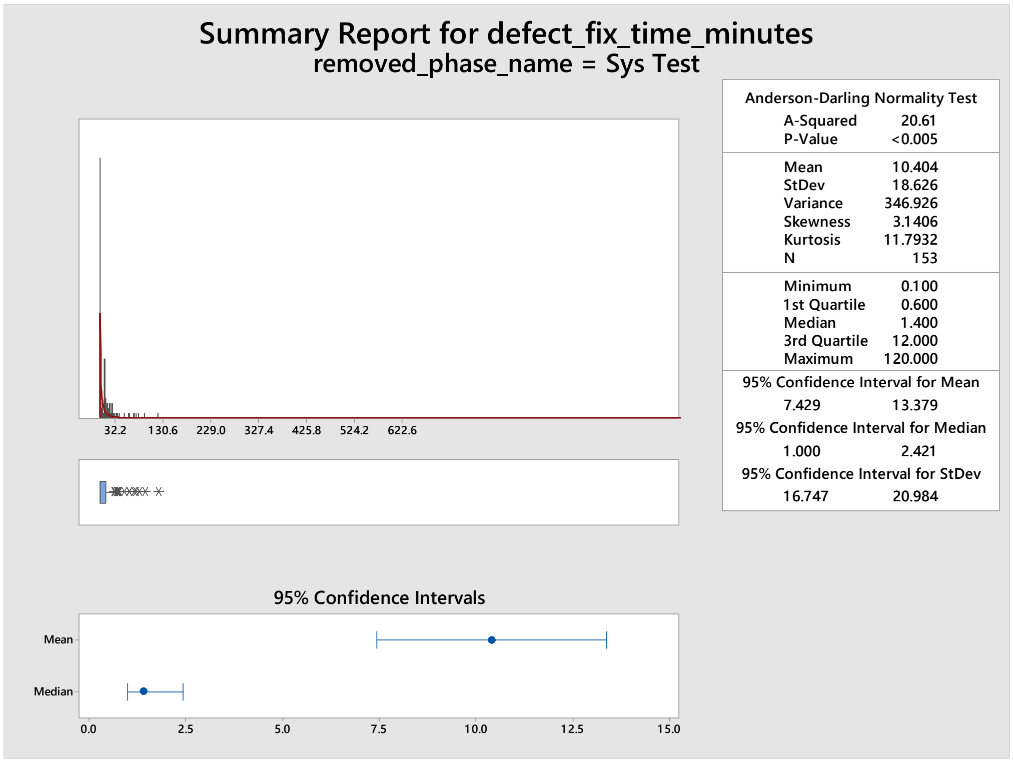


Figure 44: Defect Fix Time, System Test (Organization C)

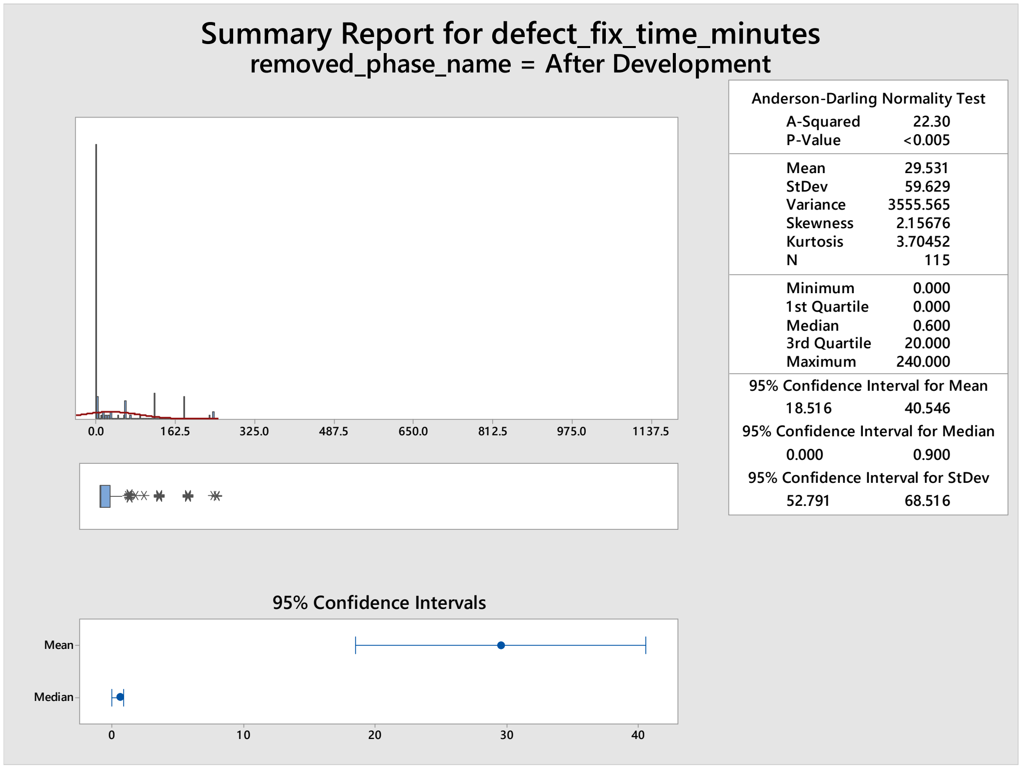
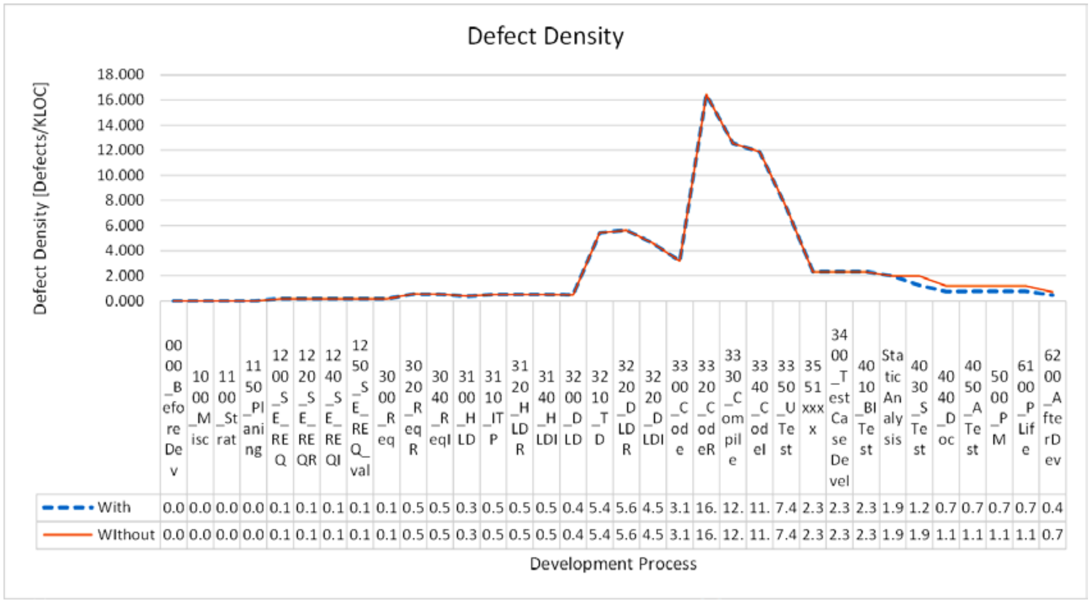


Figure 45: Defect Fix Time, After Development (Organization C)

Table 36: Average Parameters, using Static Analysis (Organization C)

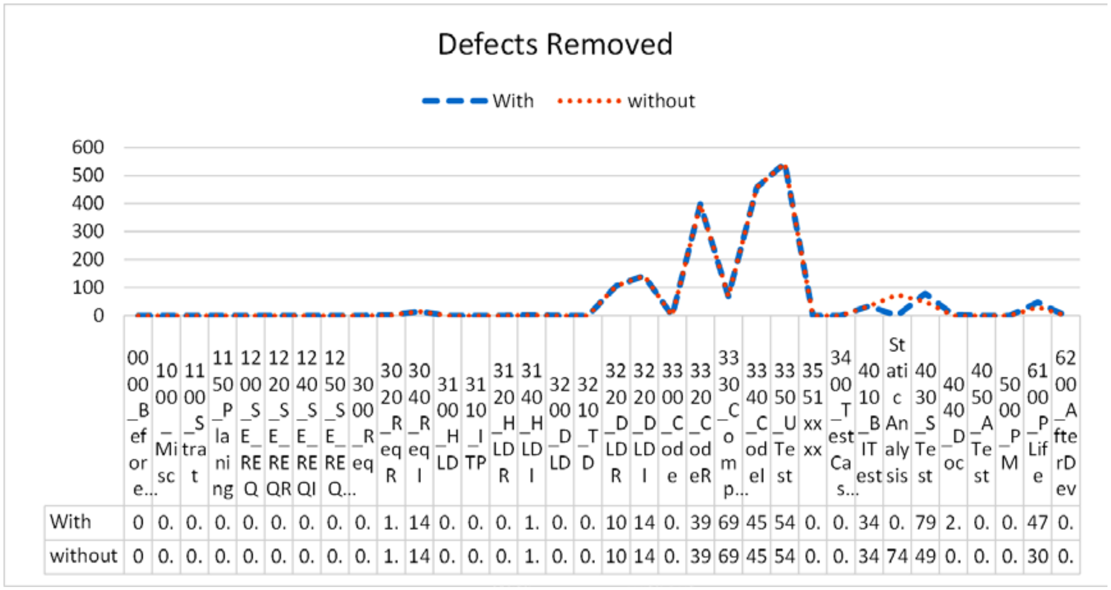
	No.Defect. Phase.Rate {LOC/Hr]	No.De-fect.Phase.Rate [Hr/LOC]	Def_Inj_Rate [Def/Hr]	Yield	FixRate [Hr/De-fect]
Req	153.1	0.0	0.1		
ReqR	1184.2	0.0	0.0	0.0308	0.04
ReqI	4439.3	0.0	0.0	0.2698	0.03
HLD	580.7	0.0	0.1		0.18
ITP	201426.2	0.0	25.2		0.04
HLDI	50356.5	0.0	0.0	0.0377	0.21
DLD	60.9	0.0	0.3		0.64
TD	466.2	0.0	0.1		0.67
DLDR	247.7	0.0	0.0	0.1856	0.19
DLDI	282.9	0.0	0.0	0.3070	0.18
Code	32.2	0.0	0.4		0.22
CodeR	126.1	0.0	0.0	0.2363	0.13
Compile	2047.5	0.0	0.1	0.0521	0.02
CodeI	161.1	0.0	0.0	0.3729	0.17
Utest	41.3	0.0	0.0	0.6881	0.32
BITest	173.2	0.0	0.0	0.1500	0.43
StaticAnalysis	392.4	0.0	0.0	0.3750	0.22
STest	174.9	0.0	0.0	0.4000	0.22
PM	1028.8	0.0	0.0		
PLife	221.2	0.0	0.0	0.4000	0.55

In this case, the model differences only occur after use of static analysis between the integration and system test phases. The test yields have a very large variation in performance, so the results are only a long-term expected average. The results suggest a modest reduction in defect density.



	0000_Before Dev	1000_Misc	1100_Strat	1150_Planing	1200_SE_REQ	1220_SE_REQR	1240_SE_REQI	1250_SE_REQ_val	3000_Req	3020_ReqR	3040_ReqI	3100_HLD	3110_ITP	3120_HLDR	3140_HLDI	3200_DLD	3210_TD	3220_DLDR	3220_DLDI	3300_Code	3320_CodeR	3330_Compile	3340_CodeI	3350_UTest	3551xxxx	3400_Test Case Devel	4010_BITest	Static Analysis	4030_STest	4040_Doc	4050_ATest	5000_PM	6100_PLife	6200_AfterDev
With	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.3	0.5	0.5	0.5	0.4	5.4	5.6	4.5	3.1	16.	12.	11.	7.4	2.3	2.3	2.3	1.9	1.2	0.7	0.7	0.7	0.7	0.4
WIthout	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	0.5	0.5	0.3	0.5	0.5	0.5	0.4	5.4	5.6	4.5	3.1	16.	12.	11.	7.4	2.3	2.3	2.3	1.9	1.9	1.1	1.1	1.1	1.1	0.7

Figure 46: Defect Density with and without Static Analysis Phase (Organization C)



	0000_Before...	1000_Misc	1100_Strat	1150_Planing	1200_SE_REQ	1220_SE_REQR	1240_SE_REQI	1250_SE_Q...	3000_Req	3020_ReqR	3040_ReqI	3100_HLD	3110_ITP	3120_HLDR	3140_HLDI	3200_DLD	3210_TD	3220_DLDR	3220_DLDI	3300_Code	3320_CodeR	3330_Comp...	3340_CodeI	3350_UTest	3551xxxx	3400_TestCas...	4010_BITest	Static Analysis	4030_STest	4040_Doc	4050_ATest	5000_PM	6100_PLife	6200_AfterDev
With	0	0.	0.	0.	0.	0.	0.	0.	0.	1.	14	0.	0.	0.	1.	0.	0.	10	14	0.	39	69	45	54	0.	0.	34	0.	79	2.	0.	0.	47	0.
without	0	0.	0.	0.	0.	0.	0.	0.	0.	1.	14	0.	0.	0.	1.	0.	0.	10	14	0.	39	69	45	54	0.	0.	34	74	49	0.	0.	0.	30	0.

Figure 47: Defect Removals by Phase (Organization C)

In this case, the defect density after test is lower, but total effort increased. This occurs because more defects were removed in test, and these defects required additional effort to repair. The find and fix time was comparable to the other test defects.

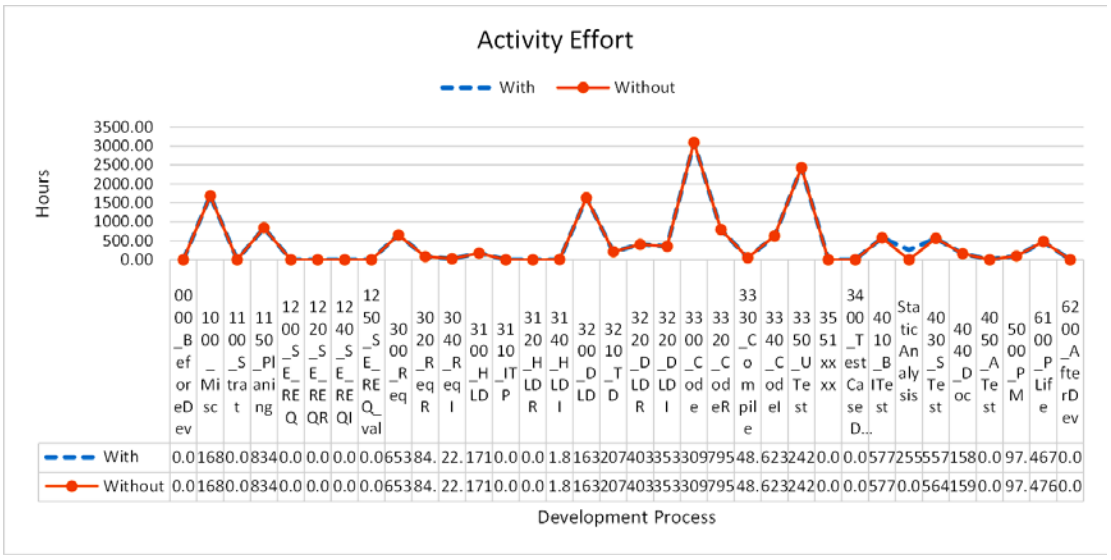


Figure 48: Cumulative Effort with and without Static Analysis

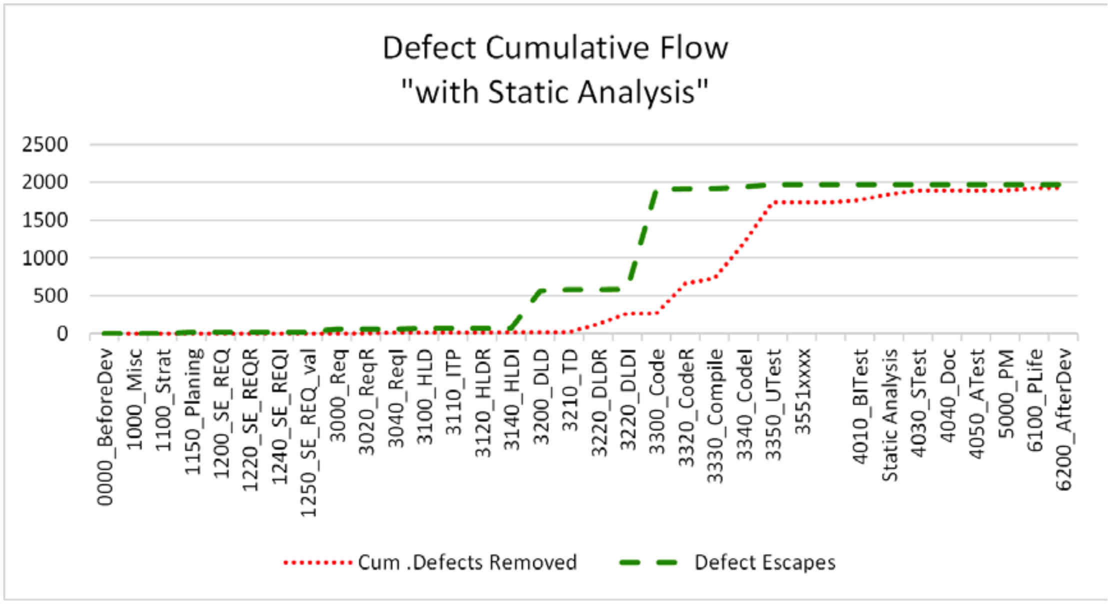


Figure 49: Cumulative Flow of Defects with Static Analysis

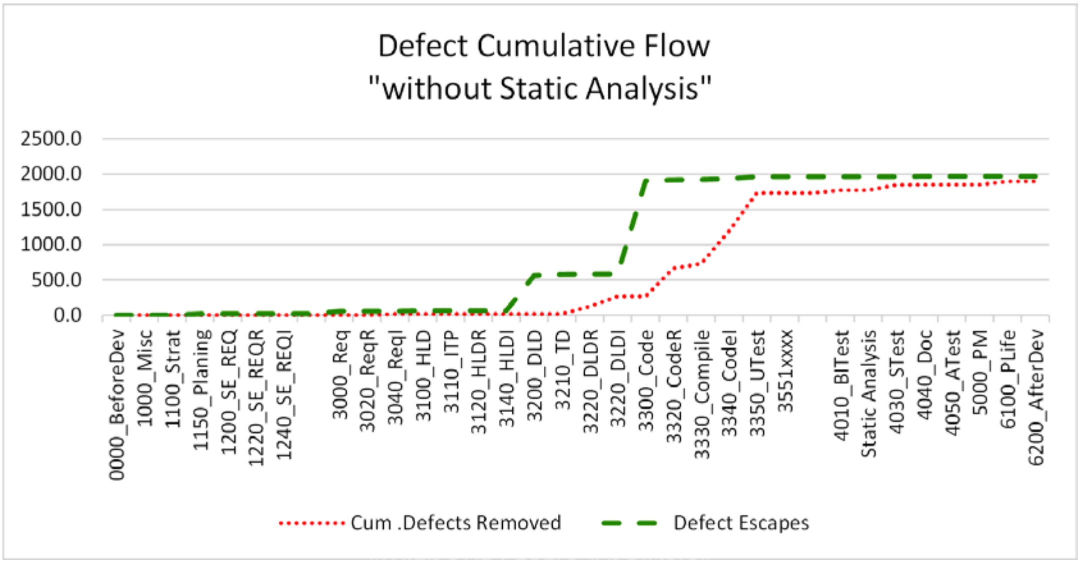


Figure 50: Defect Cumulative Flow, without Static Analysis

4.4 Overall Find and Fix Times

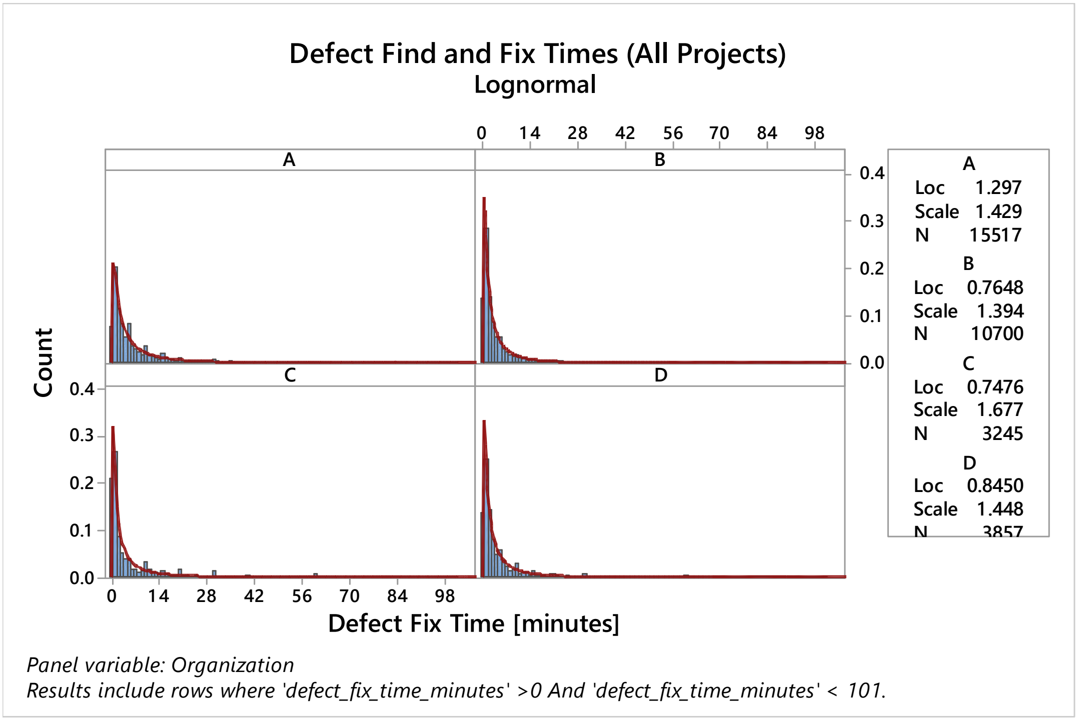


Figure 51: Organization Find and Fix Time Distributions

The organization results for defect find and fix time distributions for organizations A, B, and C are included, along with another organization (Organization D) that was not included in the study from an embedded devices domain. The vertical and horizontal scales are the same. The distributions are similar in overall range and roughly in shape. Organization A has the fewest very short find and fix times. In all organizations, the vast majority of defects were resolved in less than 15 minutes.

5 Discussion

5.1 Effectiveness of Defect Removal

Static analysis is effective, but it does not replace other quality activities. Organization A used static analysis in the code review phase and had a median removal yield of 12.4% and a range of 10.5% to 18.1%. The total number of defects removed, 2256, was 2.3 times the number removed in compile, but only about one-fourth as many defects as were removed in code inspection. Code inspection yield had a median value of 68% and a range between 54% and 76%. The high yield and relatively low rates for code inspection demonstrate that the team understood how to perform effective reviews and inspections. The high rates, rate variance, and lack of correlation between rate and yield in code review reinforce our belief that defect removal in the review phase is entirely driven by the tool, not by personal review techniques.

In context, the static analysis removal yield is only a fraction of the removal yields for code inspection, design inspection, or unit test. Moreover, the static analysis yield is lower than we would expect from a good code review. If the static analysis has crowded out personal code review, the net defect removal may have gone down. This suggests a potential risk of a so-called Peltzman effect in which compensating behavior partially offsets the benefits [Peltzman 1975].

Organization B did not isolate the tool use into a separately tracked activity as did Organization A and, to a lesser extent, Organization C. The strong correlation between code review and code inspection yields as seen in Figure 24 suggest that code reviews proceed independently.

Based on the modeling and defect labeling, we estimated a company-average reduction of 11% in escapes, which is comparable to the lower bound of Company A. Unfortunately, we also see some hints of a Peltzman effect by observing that the median review rate was 165 LOC/hr for the projects with the most static analysis finds, while the median code review rate was 359 LOC/hr for the others. Nonetheless, the median review yield was 22% for all projects and also 22% for the projects with the most static analysis finds.

A 12% reduction in defects is modest compared to the high yields in code inspection and unit test, which had yields of 67% and 60%. As with Organization A we found a modest improvement rather than a replacement for other activities.

Organization C had the smallest obvious improvement by implementing code and static analysis at build time before final test. This small scale is an artifact of the overall low levels of defects entering that phase because the yield was 63%. This high yield could be an artifact of either 1) ineffective finds of defects downstream (leading to an overestimation of the yield), or 2) the new tool finding defects that were resistant to other techniques.

In either case, the tool was clearly effective, finding 74 total defects and reducing the defect density from 1.9 to 1.2 defects/KLOC.

5.2 Cost of Defect Removal

Organization C ran the tool during the build process. Developers only remediated the issues. We looked at both time of defects and total time during acceptance test. The phase time divided by defects gives 149 minutes per defect. The logged fix time was only 535 minutes compared to 11055 minutes in that phase. The actual defect fix time was only about 7.2 minutes per defect.

The difference appears to be that the analysis time for the tool findings was considerably greater for this phase. We cannot resolve how much was identifying false positives and how much was simply separating analysis (i.e., find time) from the actual fix. For our economic analysis we included a fixed cost of running the tool based on the difference.

5.3 Are False Positives a Problem?

We have heard concerns about the level of false positives (i.e., spurious warnings) that would increase development costs without any direct benefit. Our data did not record all warnings, so we cannot directly measure the false positive rate. We did, however, measure the total developer effort in a phase and found no evidence that false positives were a problem. It may be that false positives were resolved very quickly, or that the tools were configured to reduce the incidence of false positives. The latter might also reduce overall effectiveness. Our study cannot resolve this, except to note that false positives were not a visible problem.

Static analysis cannot solve all quality or security problems. At most, static analysis tools look for a fixed set of patterns, or rules, in the code. Both false positives and false negatives (i.e., escapes) will occur. Because static analysis is prescriptive, the findings for a given situation will be consistent and depend entirely on the state of the code/binary. Future work might analyze the types of defects found in the different phases to better estimate the local rates of defect escapes.

5.4 Threats to Validity

Our analysis can be replicated if the key parameters can be measured with reasonable accuracy. Although our overall approach was designed to prefer external validity to internal validity, the results may not generalize.

5.4.1 External Validity

We cannot claim that results would be representative for all kinds of software projects. Although we included results from 39 projects, the variation of development parameters was wide among projects, even within the same company. More projects would reduce the standard errors of the parameters used. It is possible, for example, that the difference in fix times for system test and code review is much smaller than our measured averages. However, the number of projects may be constrained by a need to examine results from similar contexts. Moreover, the injection rates with an organization were much more consistent than the removal yields. Static analysis should consistently find certain types of defects if they are present. Defect injection consistency suggests that the defects will be there to be removed with any effective approach.

While we examined projects from three organizations in different development domains, an efficacy study gains strength with a broader sample. Nonetheless, this is a start on which further research should build. Another domain might have different parameters, including substantially lower static analysis yields, higher costs of static analysis remediation, higher static analysis fixed cost (e.g., higher rates of false positives or difficulty disposing of them), or much lower costs of remediation in test (coupled with high test yields). We can think of no domain where this is likely, but acknowledge the possibility that such a domain exists.

The use of TSP was a necessary condition to be included in this study; this may lead to a selection bias. TSP projects in general have a tendency to higher quality, thus biasing the results. Moreover, TSP projects already exhibit a level of process discipline that may not be expected in the general population, and this discipline may carry over to use of static analysis tools. Less than competent use of the tools may not provide overall benefits.

Our analysis approach requires data from a more or less complete project. There may be an unseen survival bias in which failed or canceled projects behave differently with respect to the static analysis.

5.4.2 Construct Validity

Construct validity assesses whether the study findings could be incorrect because of misleading data or incorrect assumptions in the model.

TSP data quality has previously been studied [Shirai 2014, Salazar 2014]. The projects that were used all contained data that passed basic required consistency checks including distributional properties and consistence among the logs. The association of defect finds with the tool was discussed with the data analysis.

We make the simplifying assumption that all defects are removed with similar effectiveness in different phases and that the finds will be more or less random. We know different types of activities remove defect types at different rates [Vallespir 2011, 2012]. Addressing this threat remains for future work.

The model also uses average values for the organizations. Mathematically, the project parameters will reproduce, ex post, the project results. In practice, parameters vary both statistically and because of process drift. With sufficient data, the model could be extended to use distributions to produce probability ranges for the outcomes. This remains for future work.

We measure the defect findings, but do not directly measure fielded faults, severity, or association of the defects with security. We assume relationships found in prior work [Woody 2015, Emanuelsson 2008, Wheeler 2016, Chulani 1999].

5.4.3 Internal Validity

Internal validity evaluates whether causal findings are due to factors that have not been controlled or measured. Causality, by design, is beyond the scope of this work because attempts to control the behavior would weaken external validity. Instead, we assume a causal system and attempt to measure the effects of the tools as implemented in real-world environments. Internal validity could suffer to the extent that the assumptions were violated by subjective judgments or by the participants or the analyst.

Under everyday real-world development settings, factors such as developer or management preferences, process compliance, education, training, and so forth will affect the results. Although some of these factors could be accounted for with additional observation or surveys, these were beyond the scope of this work. Blinding the analyst was impractical. The effect is minimized by minimizing subjective judgment.

5.5 Limitations and Future Work

Our effectiveness analysis is limited to the defect removal of the tools in the project context and the operational costs. A more complete treatment of cost/benefit would include acquisition and licensing costs of the tools, the cost of fielded defects, and the external costs such as impact on the user. Given additional cost information, a more complete treatment can be found in [Zheng 2006].

In this study we analyzed the operational cost and effectiveness of applying the tools in a real-world setting. We did not investigate ways to use the tools more effectively or assess the individual tool use for compliance, specific usage, or comparative effectiveness with tools of similar or dissimilar type.

We did not investigate the specific coverage of tools with respect to defect type or defect severity; therefore we cannot assess potential secondary effects when multiple tools are combined. We did, however, demonstrate that in principle it is possible to gather data using orthogonal defect categorization, and this could be used to extend the analysis capabilities. While the tool yields remain modest, it seems likely that interactions will be secondary, possibly even for similar tool types.

We investigated the cost effectiveness in organizations that already had defect escape levels that are low by industry standards [Jones 2011]. Effectiveness for lesser quality development processes was not demonstrated. In principle, our model should apply equally well to other organizations; however, the study may be inhibited by lower quality data. Our model suggests settings with less effective quality practices should benefit more from these tools because the absolute find rates will be higher and the potential reduction of test time will be greater.

The sample size is not large enough to exclude the possibility of circumstances that would make the tools very effective or very ineffective. Absence of evidence is not evidence of absence. Studies with substantially larger samples are possible in principle but face practical difficulties. We provide some recommendations for instrumentation that will make the gathering and analysis more consistent and analyzable.

Because the effect of individual tools can be modest in either relative or absolute terms, observational studies with less detailed process data will be challenged to resolve the effects on defect escapes. This will become more pronounced as additional tools are included in the development. Inconsistencies in which kinds of defects are counted in which of the development phases, and inconsistencies in effort accounting, will introduce threats to construct validity.

Following products for a longer term to track maintenance effort, deployed defects and vulnerabilities, patch deployment costs, external costs to users, and so forth will enhance external validity.

In summary, more study is needed, but the study will require improvements to the process instrumentation and data collection.

5.6 Implications for Cost-Effective Process Composition

Each organization's use of the tools was modestly effective. There seems little doubt that even software development projects that are performing well can improve code quality by including static analysis tools in the development process. Moreover, using these static analysis tools early is modestly effective in reducing development time by reducing defect find and fix in test. The difference in find and fix phase efforts more than compensated for the time to operate the tools and find/fix in test. A partial exception was when the tools were run at integration. The find/fix advantage is reduced and disappears when we consider the fix cost of evaluating the warnings.

The tools were applied inconsistently not only between organizations but between projects within the same organization. This could lead not only to inconsistent tool effectiveness, but also to difficulties in evaluating cost effectiveness. We recommend that projects adopt practices that enhance the consistency of tool use and measurement rather than leave each project to adopt an ad hoc approach.

Developers and project managers were unwilling to commit resources to either measuring or automating tool use for this study. No organization had a clear understanding of the cost effectiveness of the tools. When we consider the modest effects of the tools on the process, we are concerned that the cost/benefit trade-offs will not be apparent to management or software acquirers. We therefore recommend the automation of tool usage when practicable to improve both usage consistency and measurement. Isolating tool usage was a mechanism that was helpful with the accounting. Building automation into the build process helped Organization C institutionalize use.

We recommend the following actions to establish usage standards:

- Define the overall workflow and specify when in the development process the tools should be applied.
- Document the settings to be applied.
- Set clear guidelines for when issues must be remediated, mitigated, or ignored.

We recommend the following actions to establish measurement standards:

- Count all issues reported and mitigated.
- Account for all effort expended running the tool and resolving the issues.

Apply the tools after other manual practices. Tools were observed to sometimes crowd out manual activities. In particular, we observed a static analysis tool replacing the personal review. The teams that segregated tool use to after development, reviews, and inspection, but before test, avoided this problem.

We recommend using static checking tools before test, but after manual review and inspection for the following reasons:

- Defects are removed when the costs are less than in test.
- It avoids crowding out other effective removal techniques.
- Objective measures of the defect escapes from earlier removal activities are provided.

Defects found by static binary checkers and dynamic checkers have comparable find/fix effort to other system test defects. Where test resources are constrained (i.e., by personnel availability, the need for special equipment, or by large and time consuming regression tests), it makes the most sense to run the checkers prior to test. However, if these factors do not apply, the binary and dynamic checkers could potentially be used to validate test completeness.

Finally, we observed that there are two simple ways to get through test quickly:

1. Run ineffective tests (or no tests at all).
2. Make sure that the code being tested has few or no bugs to be found.

Software quality and security checking tools should not replace reviews and tests (even unintentionally). Instead, the processes should be composed to verify and validate that the development process is effective.

6 Conclusions

In this research we examined data from 39 software projects in three organizations and application domains that applied static analysis on code or code and binary, looking for evidence showing how static analysis affects development effort and quality. We gathered the historical process development parameters from the projects' data logs and analyzed how sensitive cost and defect escapes were to changes in the parameters that are affected by static analysis tools. Although organizations applied different tools at different times during the development process, the results are consistent with the conclusion that static analysis modestly improves the delivered quality, and presumably reduces escaped vulnerabilities.

Our analysis was based only on the operational costs; we did not consider the cost of software licenses or training costs.

We reached the following conclusions:

1. The tools studied have a positive, but modest, effect on reducing defect escapes.
2. When applied to the code or code and binary prior to integration, the tools studied also have a modest positive effect on reducing overall project costs.
3. When applied to the code and binary at or after integration, costs increased slightly but were consistent with the costs of remediating the found defects.
4. Defect fix effort from static code analysis was comparable to personal review and inspection defect fix times.
5. Defect fix times from static binary analysis were closer to defect fix times from system test.
6. Static code analysis tools are associated with reduced overall development time because they reduce the number of defects found in the more expensive test phases.
7. Static binary analyses do not shorten development time and require a substantial effort to address the findings. The defect fix costs are similar to system test.

We found that defect fix effort was similar to that of defects found in inspections, and binary analysis results were between code inspection and late test. We therefore observe that overall development time is reduced when the defects are removed earlier in the process, but effort is not reduced when the tools are applied later.

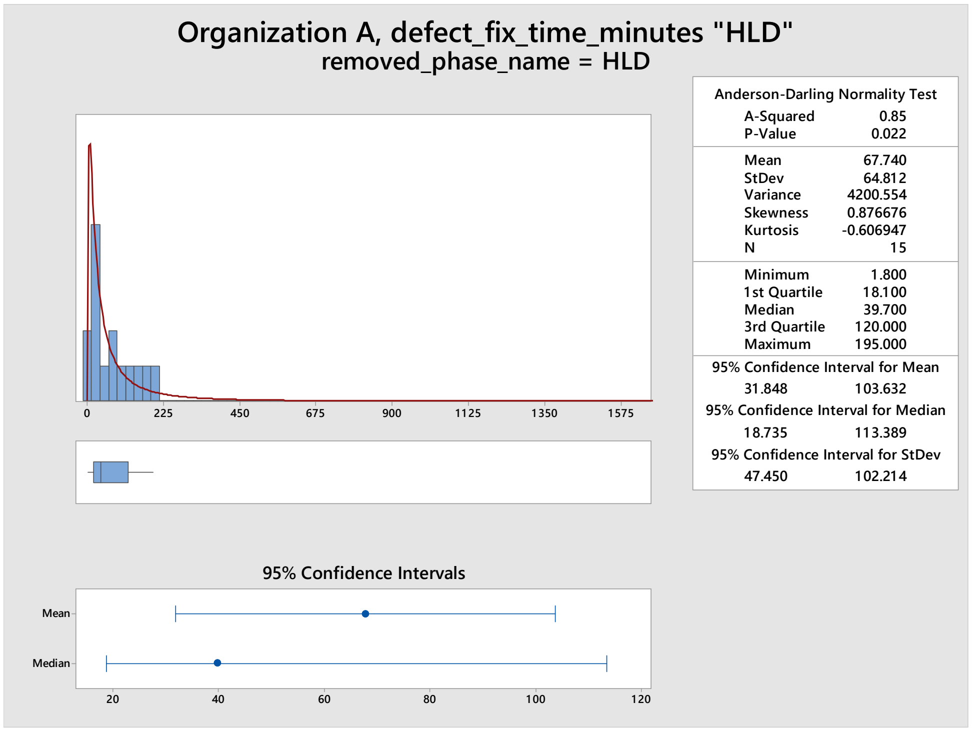
Because static analysis finds were only a small portion the total defects, we also conclude that static analysis alone is not sufficient, but is a cost effective, incremental improvement to development that should be used with other defect removal techniques. This raises the question of why static analysis is not universally practiced and what incentives will be necessary to increase utilization. One potential answer to the usage gap may be that while the benefits are real but modest, without detailed effort and defect accounting they may be obscured by natural variation in the software development process. A second answer may be that the costs are very visible, while the benefits (i.e., reductions in downstream defects) are less apparent because they are delayed in time and are sometimes observed by a different set of developers.

Another observation about the results of using the tools is that the benefits are challenging to connect directly to effort, quality, or development duration. Despite similar data gathering and process phases, these three organizations implemented the tools somewhat differently. Understanding the context and specific usage was a major challenge during our research and required much additional background information, including discussions with participants and the use of project records. This approach will not scale to larger studies. More standardization in the accounting and automation could enable studies with more data and a broader set of domains.

Increased automation of static analysis and data collection may help to mitigate another problem we observed. At least one organization appears to have more or less ceased to perform personal reviews of the code, depending instead on the static analysis tool. This suggests that there is a risk that the benefits may be partially offset by developers relying on the tool and foregoing or superficially executing other effective defect removal techniques. The other two organizations at least partially automated the static analysis into the build later in the development. The organization that most fully automated the tool had no obvious problems with process compliance.

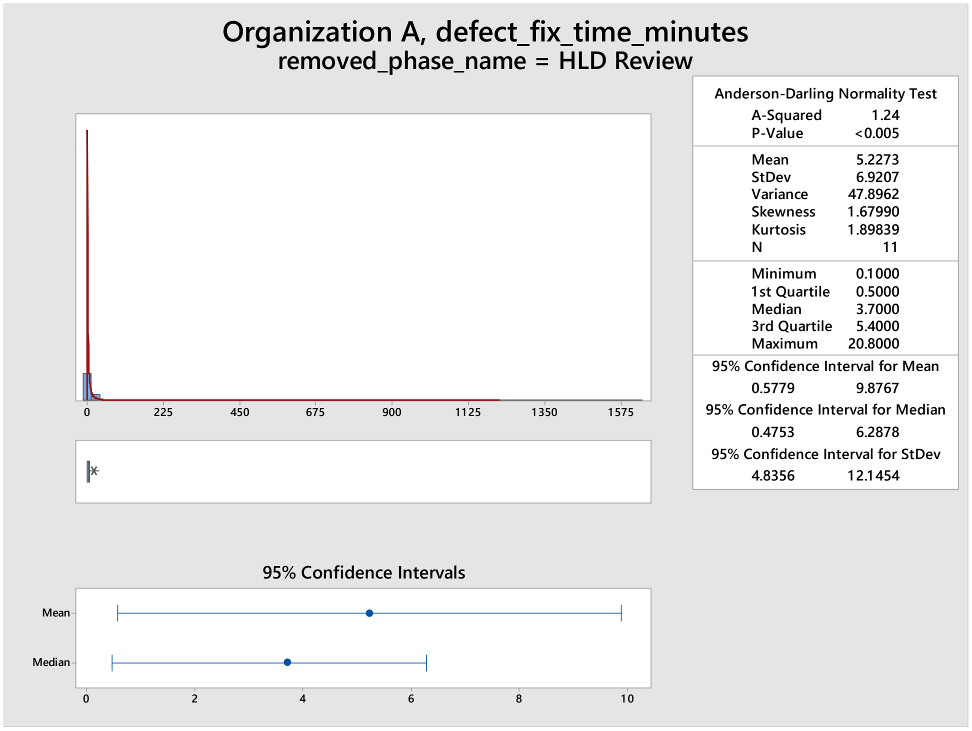
Finally, continued progress of software analytics can make economic analyses (such as those in this report) more cost-effective and feasible. Although global rules may apply, examination of local behaviors is of significant benefit.

Appendix: Additional Data



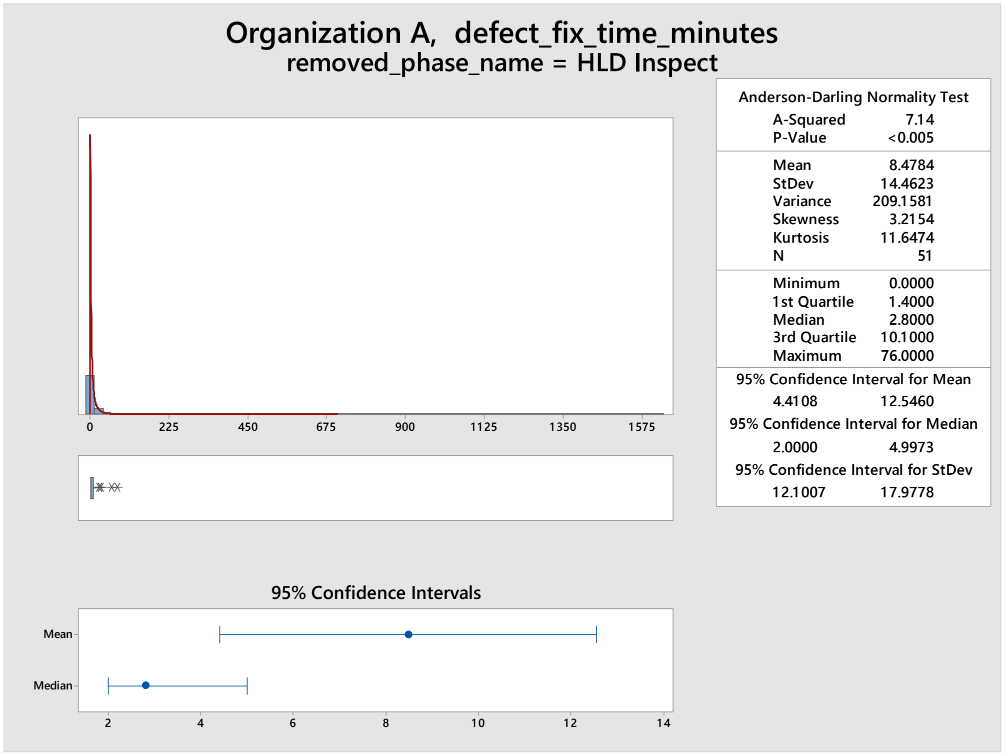
Anderson-Darling Normality Test	
A-Squared	0.85
P-Value	0.022
Mean	67.740
StDev	64.812
Variance	4200.554
Skewness	0.876676
Kurtosis	-0.606947
N	15
Minimum	1.800
1st Quartile	18.100
Median	39.700
3rd Quartile	120.000
Maximum	195.000
95% Confidence Interval for Mean	
31.848	103.632
95% Confidence Interval for Median	
18.735	113.389
95% Confidence Interval for StDev	
47.450	102.214

Figure 52: Defect Distribution, HLD (Organization A)



Anderson-Darling Normality Test	
A-Squared	1.24
P-Value	<0.005
Mean	5.2273
StDev	6.9207
Variance	47.8962
Skewness	1.67990
Kurtosis	1.89839
N	11
Minimum	0.1000
1st Quartile	0.5000
Median	3.7000
3rd Quartile	5.4000
Maximum	20.8000
95% Confidence Interval for Mean	
0.5779	9.8767
95% Confidence Interval for Median	
0.4753	6.2878
95% Confidence Interval for StDev	
4.8356	12.1454

Figure 53: Defect Distribution, HLD Review (Organization A)



Anderson-Darling Normality Test

A-Squared	7.14
P-Value	<0.005
Mean	8.4784
StDev	14.4623
Variance	209.1581
Skewness	3.2154
Kurtosis	11.6474
N	51
Minimum	0.0000
1st Quartile	1.4000
Median	2.8000
3rd Quartile	10.1000
Maximum	76.0000

95% Confidence Interval for Mean

4.4108 12.5460

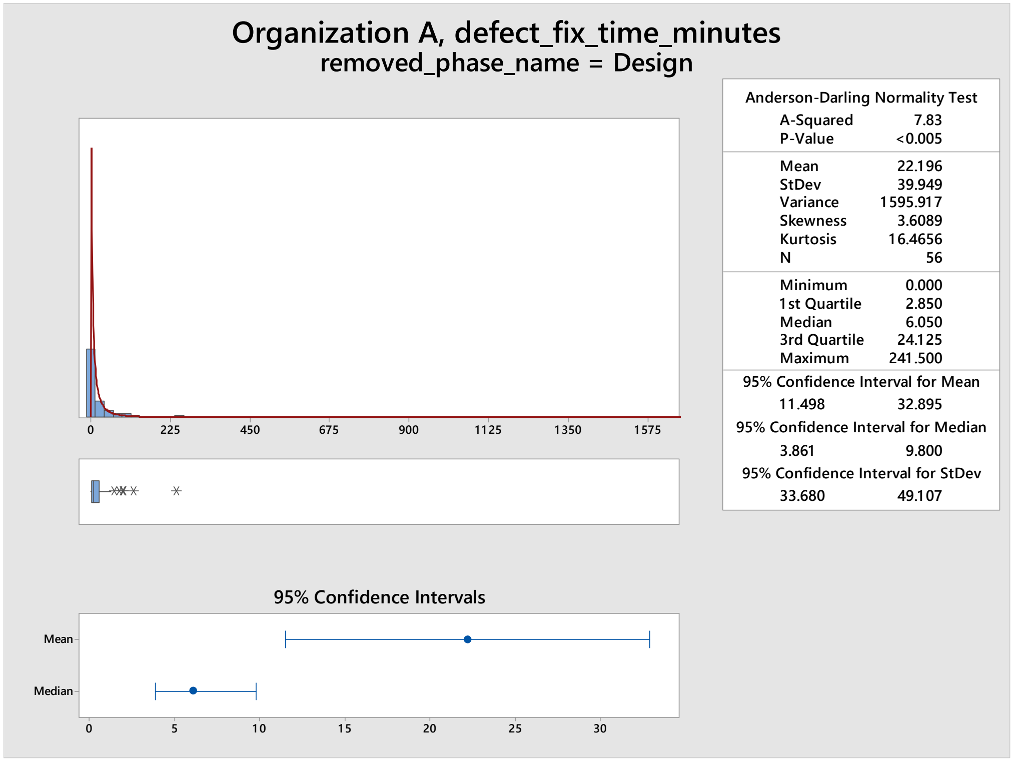
95% Confidence Interval for Median

2.0000 4.9973

95% Confidence Interval for StDev

12.1007 17.9778

Figure 54: Defect Distribution, HLD Inspect (Organization A)



Anderson-Darling Normality Test

A-Squared	7.83
P-Value	<0.005
Mean	22.196
StDev	39.949
Variance	1595.917
Skewness	3.6089
Kurtosis	16.4656
N	56
Minimum	0.000
1st Quartile	2.850
Median	6.050
3rd Quartile	24.125
Maximum	241.500

95% Confidence Interval for Mean

11.498 32.895

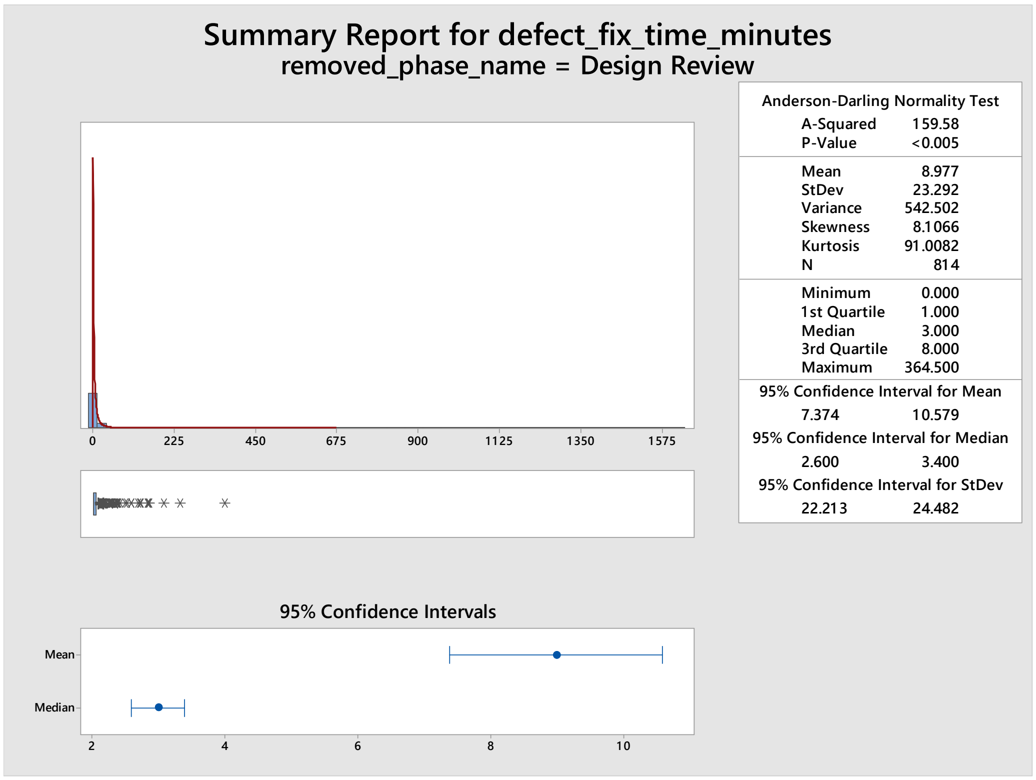
95% Confidence Interval for Median

3.861 9.800

95% Confidence Interval for StDev

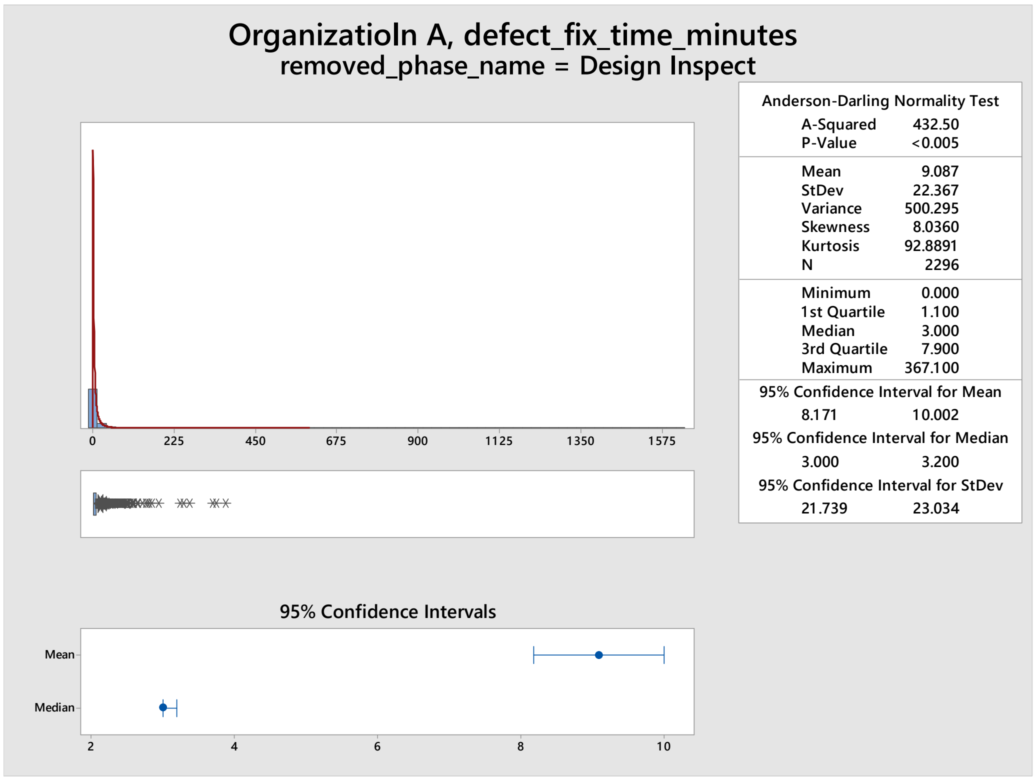
33.680 49.107

Figure 55: Defect Distribution, Design (Organization A)



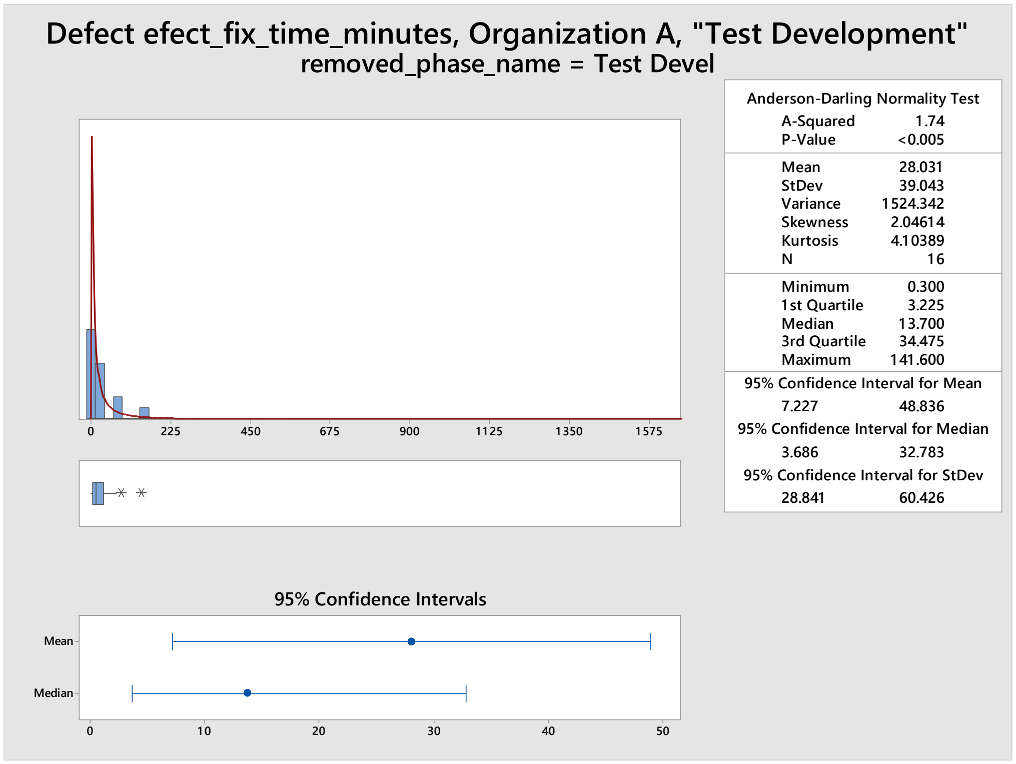
Anderson-Darling Normality Test	
A-Squared	159.58
P-Value	<0.005
Mean	8.977
StDev	23.292
Variance	542.502
Skewness	8.1066
Kurtosis	91.0082
N	814
Minimum	0.000
1st Quartile	1.000
Median	3.000
3rd Quartile	8.000
Maximum	364.500
95% Confidence Interval for Mean	
7.374	10.579
95% Confidence Interval for Median	
2.600	3.400
95% Confidence Interval for StDev	
22.213	24.482

Figure 56: Defect Distribution, Design Review (Organization A)



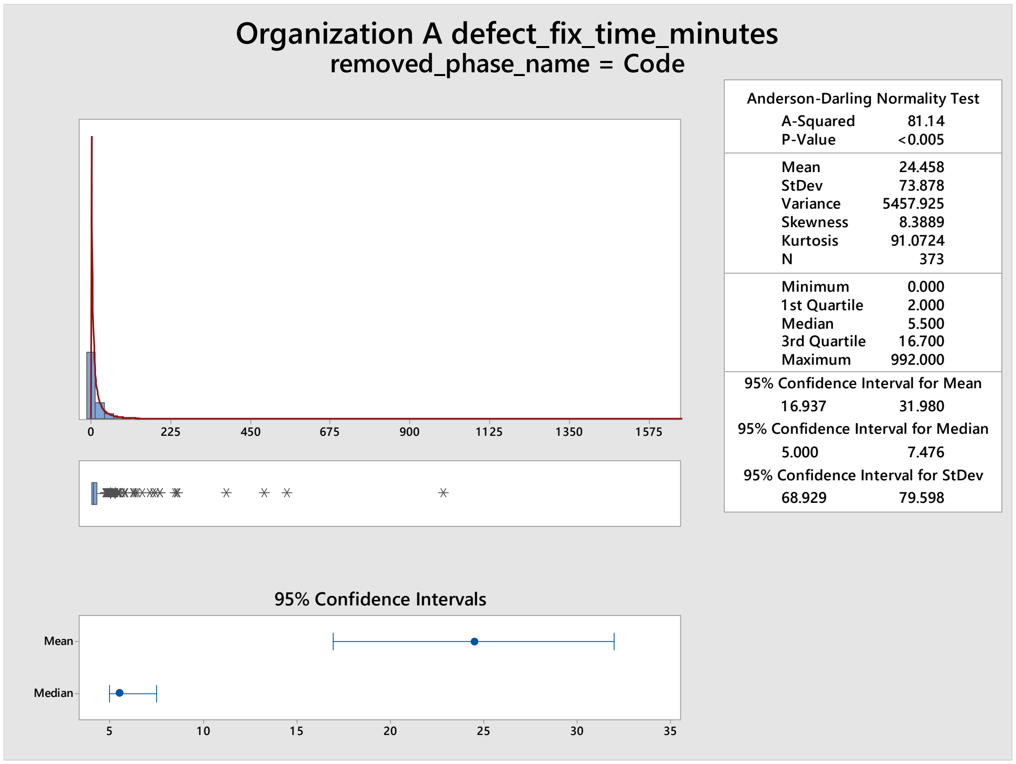
Anderson-Darling Normality Test	
A-Squared	432.50
P-Value	<0.005
Mean	9.087
StDev	22.367
Variance	500.295
Skewness	8.0360
Kurtosis	92.8891
N	2296
Minimum	0.000
1st Quartile	1.100
Median	3.000
3rd Quartile	7.900
Maximum	367.100
95% Confidence Interval for Mean	
8.171	10.002
95% Confidence Interval for Median	
3.000	3.200
95% Confidence Interval for StDev	
21.739	23.034

Figure 57: Defect Distribution, Design Inspect (Organization A)



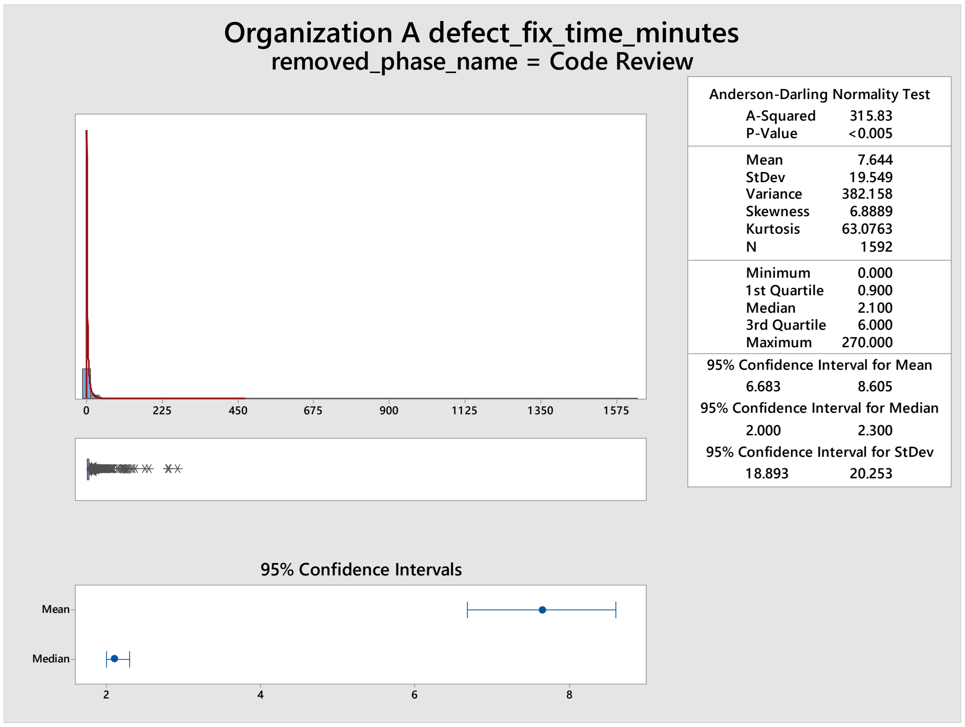
Anderson-Darling Normality Test	
A-Squared	1.74
P-Value	<0.005
Mean	28.031
StDev	39.043
Variance	1524.342
Skewness	2.04614
Kurtosis	4.10389
N	16
Minimum	0.300
1st Quartile	3.225
Median	13.700
3rd Quartile	34.475
Maximum	141.600
95% Confidence Interval for Mean	
7.227	48.836
95% Confidence Interval for Median	
3.686	32.783
95% Confidence Interval for StDev	
28.841	60.426

Figure 58: Defect Distribution, Test Development (Organization A)



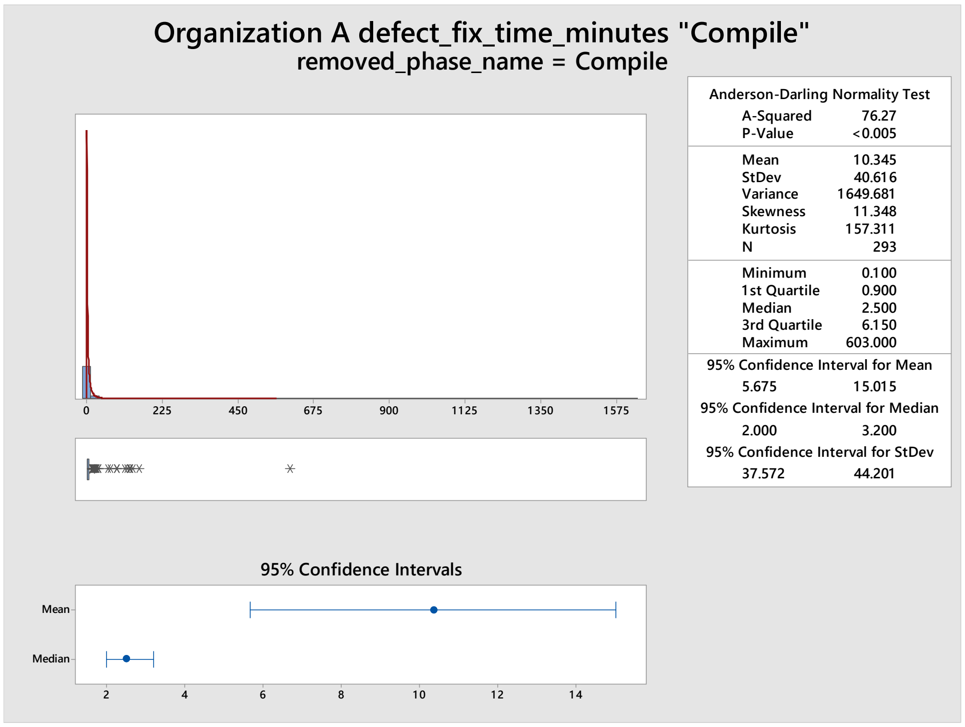
Anderson-Darling Normality Test	
A-Squared	81.14
P-Value	<0.005
Mean	24.458
StDev	73.878
Variance	5457.925
Skewness	8.3889
Kurtosis	91.0724
N	373
Minimum	0.000
1st Quartile	2.000
Median	5.500
3rd Quartile	16.700
Maximum	992.000
95% Confidence Interval for Mean	
16.937	31.980
95% Confidence Interval for Median	
5.000	7.476
95% Confidence Interval for StDev	
68.929	79.598

Figure 59: Defect Distribution, Code (Organization A)



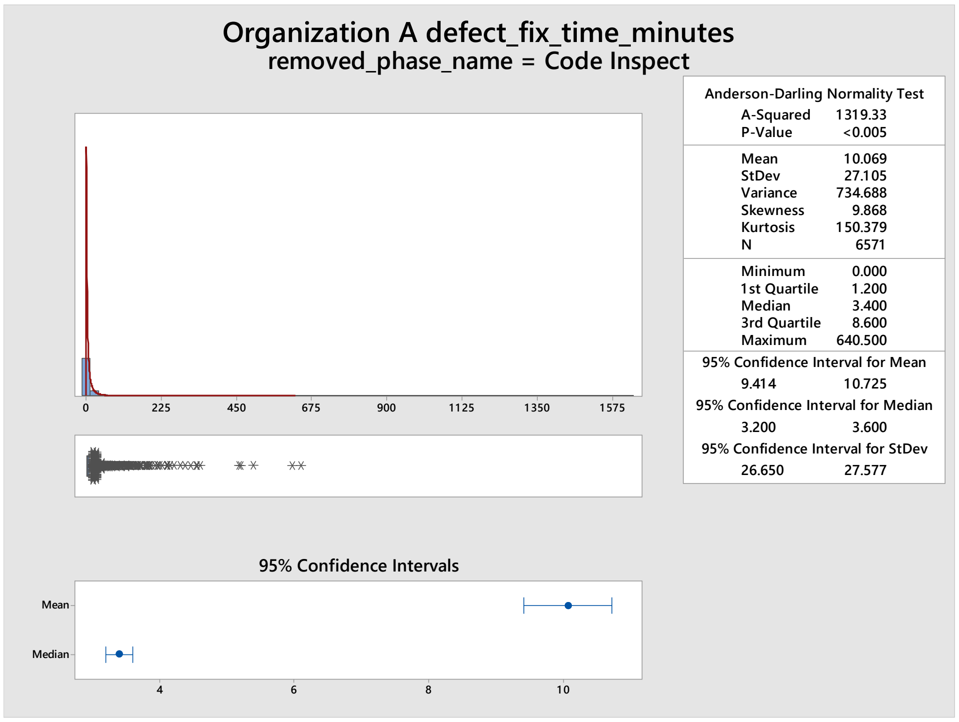
Anderson-Darling Normality Test	
A-Squared	315.83
P-Value	<0.005
Mean	7.644
StDev	19.549
Variance	382.158
Skewness	6.8889
Kurtosis	63.0763
N	1592
Minimum	0.000
1st Quartile	0.900
Median	2.100
3rd Quartile	6.000
Maximum	270.000
95% Confidence Interval for Mean	
6.683	8.605
95% Confidence Interval for Median	
2.000	2.300
95% Confidence Interval for StDev	
18.893	20.253

Figure 60: Defect Distribution, Code Review (Organization A)



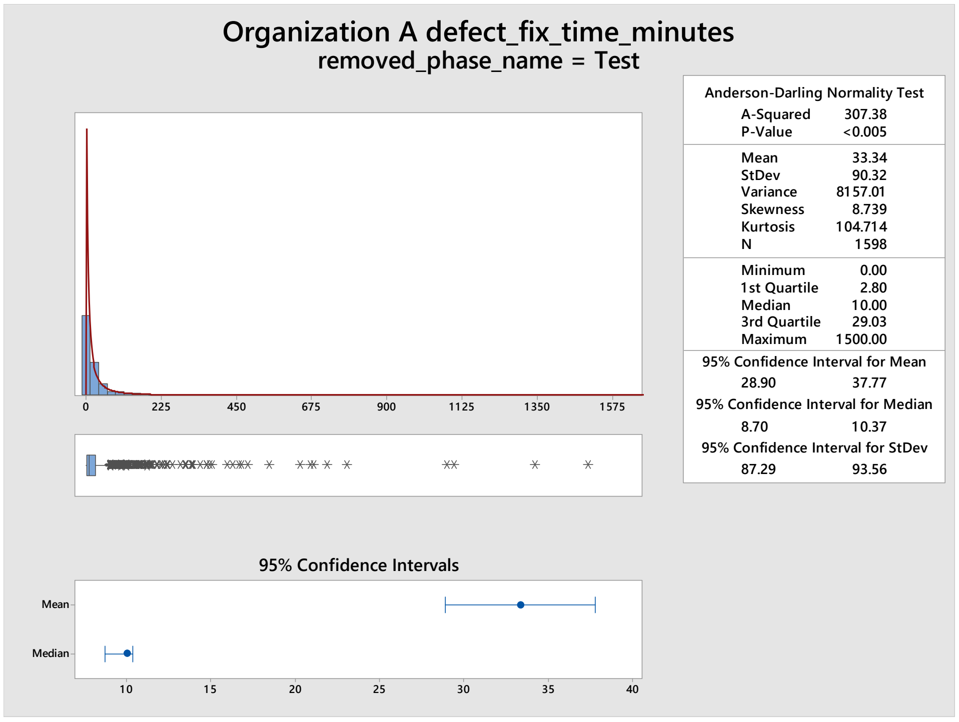
Anderson-Darling Normality Test	
A-Squared	76.27
P-Value	<0.005
Mean	10.345
StDev	40.616
Variance	1649.681
Skewness	11.348
Kurtosis	157.311
N	293
Minimum	0.100
1st Quartile	0.900
Median	2.500
3rd Quartile	6.150
Maximum	603.000
95% Confidence Interval for Mean	
5.675	15.015
95% Confidence Interval for Median	
2.000	3.200
95% Confidence Interval for StDev	
37.572	44.201

Figure 61: Defect Distribution, Compile (Organization A)



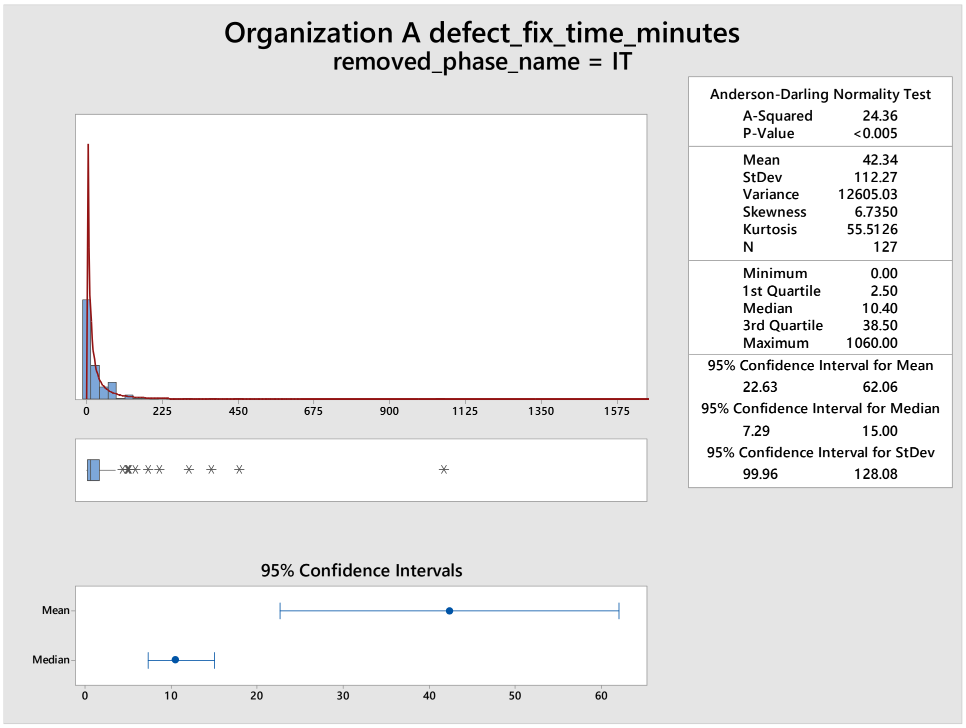
Anderson-Darling Normality Test	
A-Squared	1319.33
P-Value	<0.005
Mean	10.069
StDev	27.105
Variance	734.688
Skewness	9.868
Kurtosis	150.379
N	6571
Minimum	0.000
1st Quartile	1.200
Median	3.400
3rd Quartile	8.600
Maximum	640.500
95% Confidence Interval for Mean	
9.414	10.725
95% Confidence Interval for Median	
3.200	3.600
95% Confidence Interval for StDev	
26.650	27.577

Figure 62: Defect Distribution, Code Inspect (Organization A)



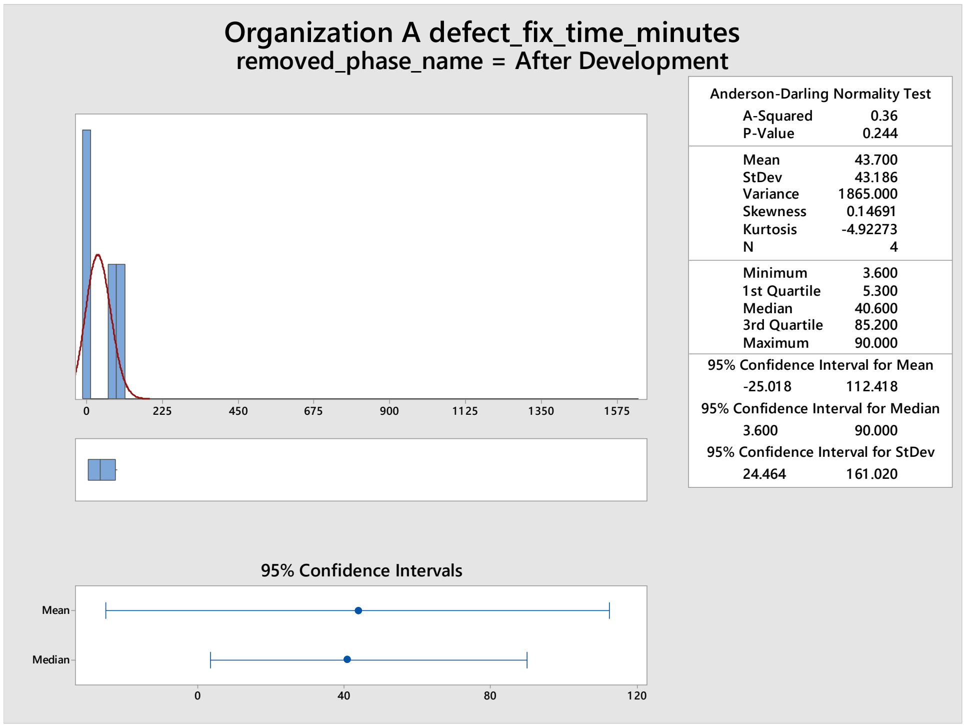
Anderson-Darling Normality Test	
A-Squared	307.38
P-Value	<0.005
Mean	33.34
StDev	90.32
Variance	8157.01
Skewness	8.739
Kurtosis	104.714
N	1598
Minimum	0.00
1st Quartile	2.80
Median	10.00
3rd Quartile	29.03
Maximum	1500.00
95% Confidence Interval for Mean	
28.90	37.77
95% Confidence Interval for Median	
8.70	10.37
95% Confidence Interval for StDev	
87.29	93.56

Figure 63: Unit Test Defect Distribution, Test (Organization A)



Anderson-Darling Normality Test	
A-Squared	24.36
P-Value	<0.005
Mean	42.34
StDev	112.27
Variance	12605.03
Skewness	6.7350
Kurtosis	55.5126
N	127
Minimum	0.00
1st Quartile	2.50
Median	10.40
3rd Quartile	38.50
Maximum	1060.00
95% Confidence Interval for Mean	
22.63	62.06
95% Confidence Interval for Median	
7.29	15.00
95% Confidence Interval for StDev	
99.96	128.08

Figure 64: Defect Distribution, Integration Test (Organization A)



Anderson-Darling Normality Test	
A-Squared	0.36
P-Value	0.244
Mean	43.700
StDev	43.186
Variance	1865.000
Skewness	0.14691
Kurtosis	-4.92273
N	4
Minimum	3.600
1st Quartile	5.300
Median	40.600
3rd Quartile	85.200
Maximum	90.000
95% Confidence Interval for Mean	
-25.018	112.418
95% Confidence Interval for Median	
3.600	90.000
95% Confidence Interval for StDev	
24.464	161.020

Figure 65: Defect Distribution, After Development (Organization A)

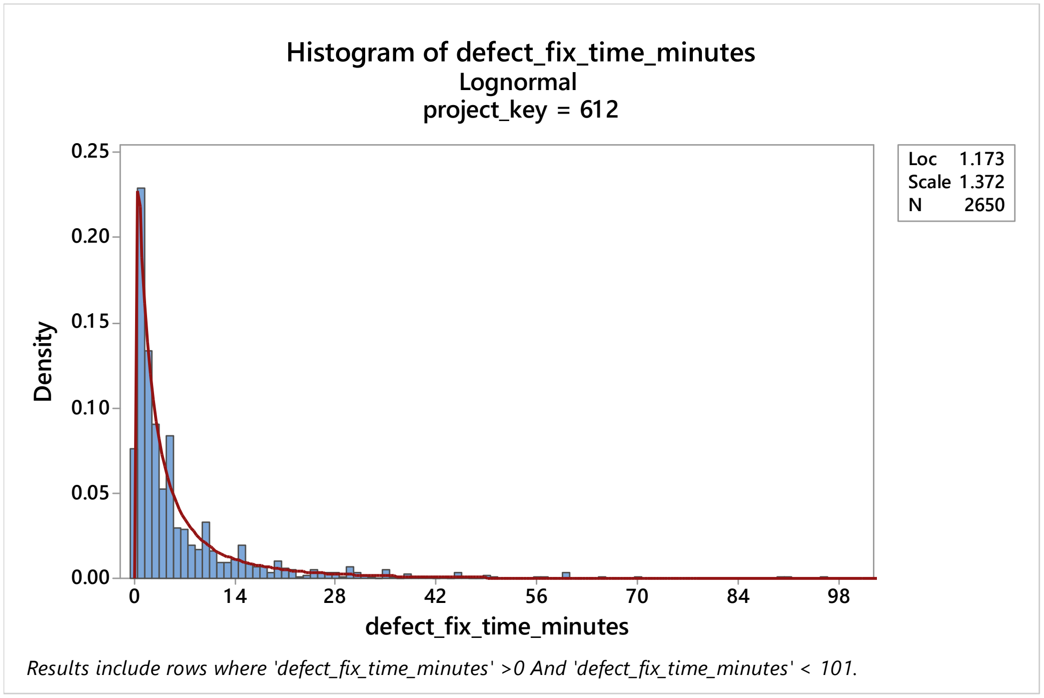


Figure 66: Defect Fix Time Distribution, Project 612 (Organization A)

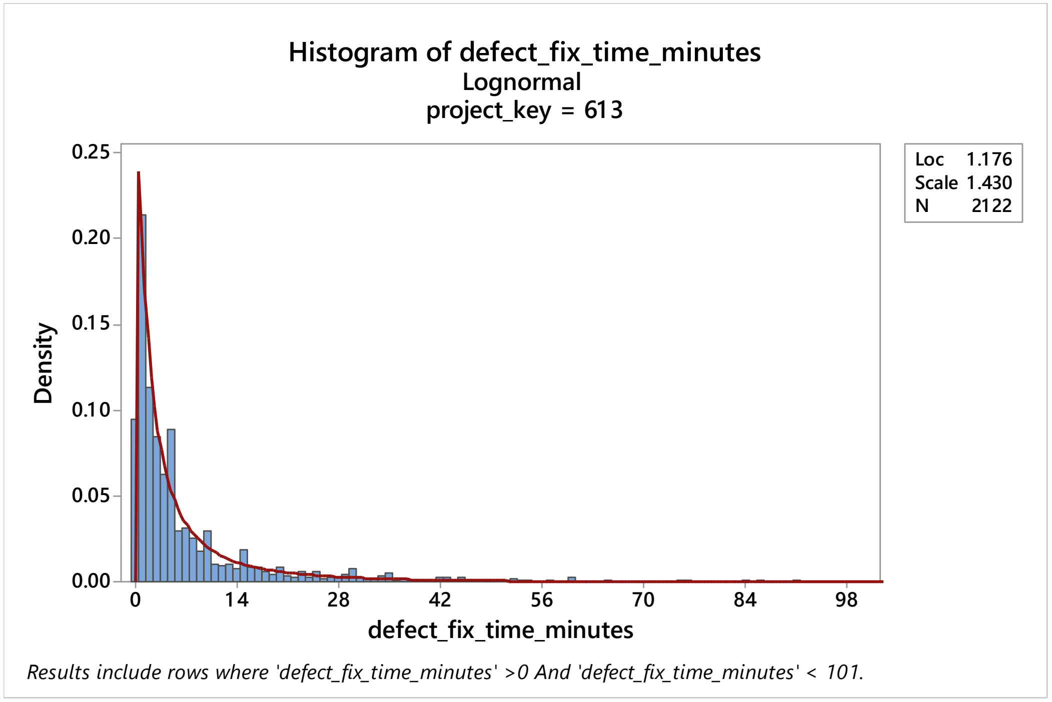


Figure 67: Defect Fix Time Distribution, Project 613 (Organization A)

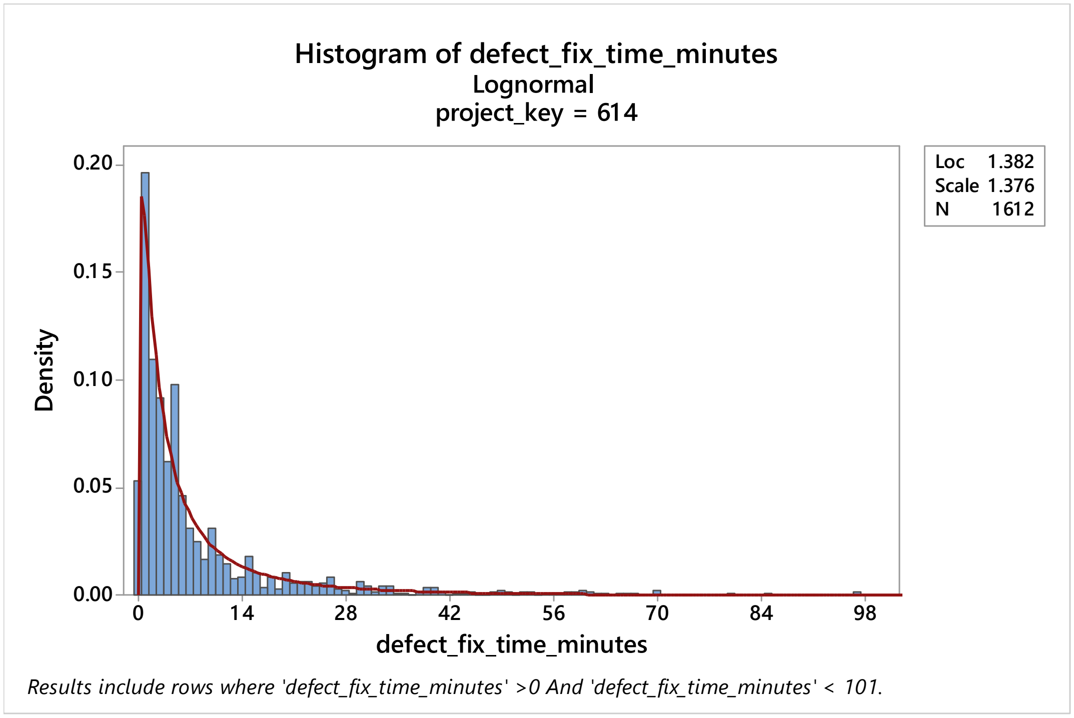


Figure 68: Defect Fix Time Distribution, Project 614 (Organization A)

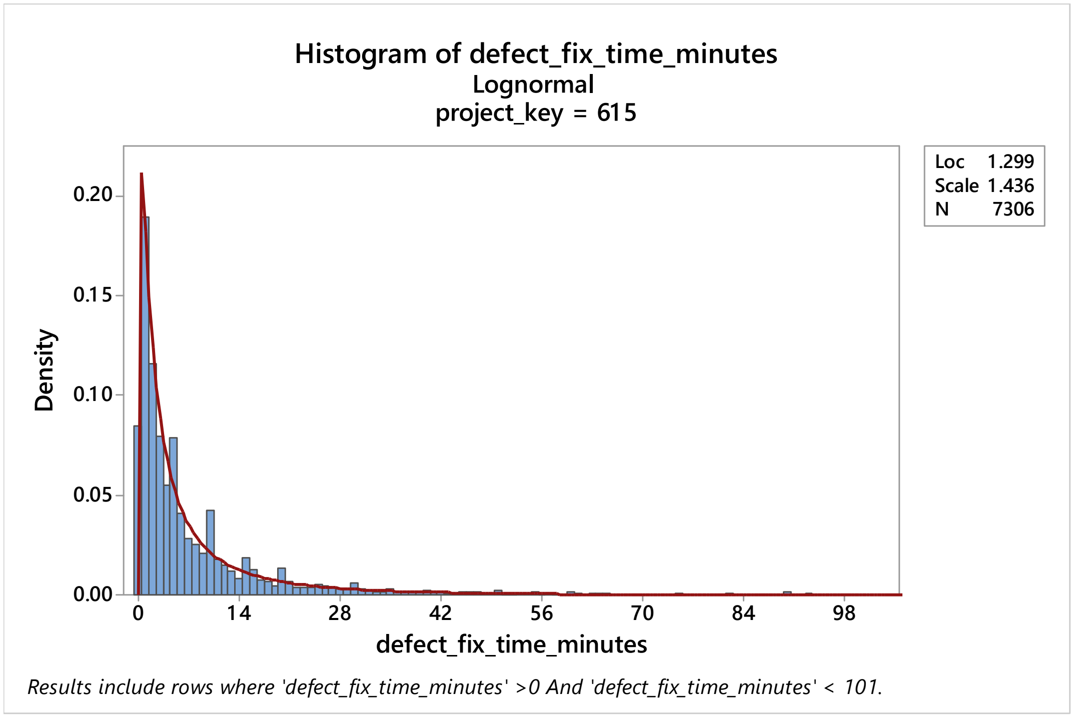


Figure 69: Defect Fix Effort, Project 615 (Organization A)

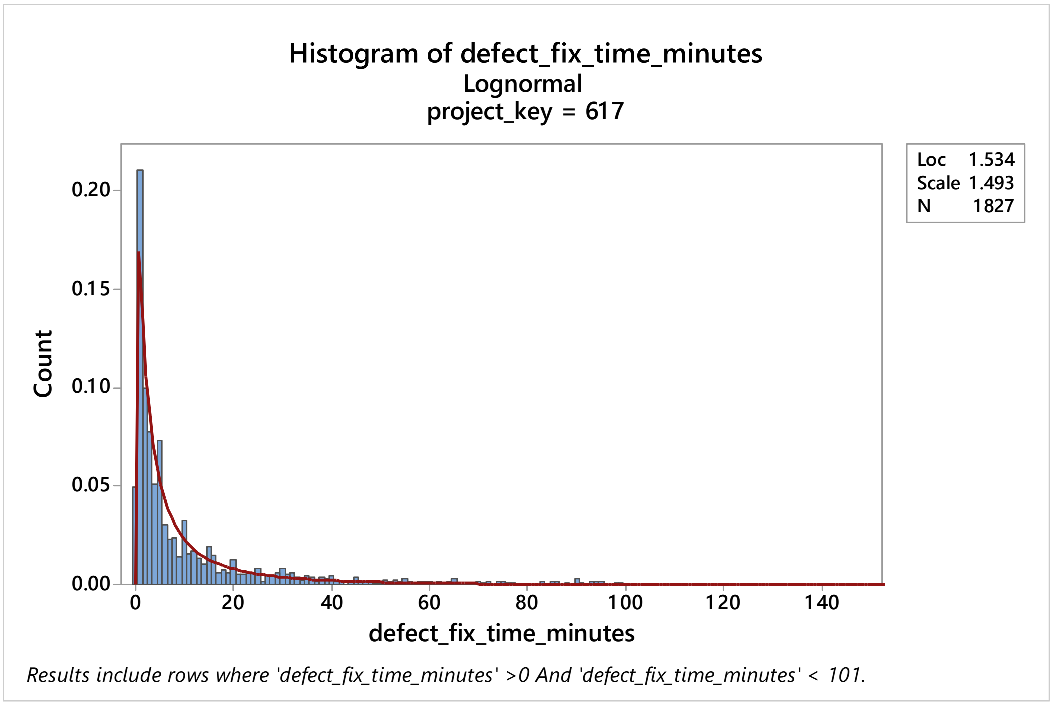


Figure 70: Defect Fix Effort, Project 617 (Organization A)

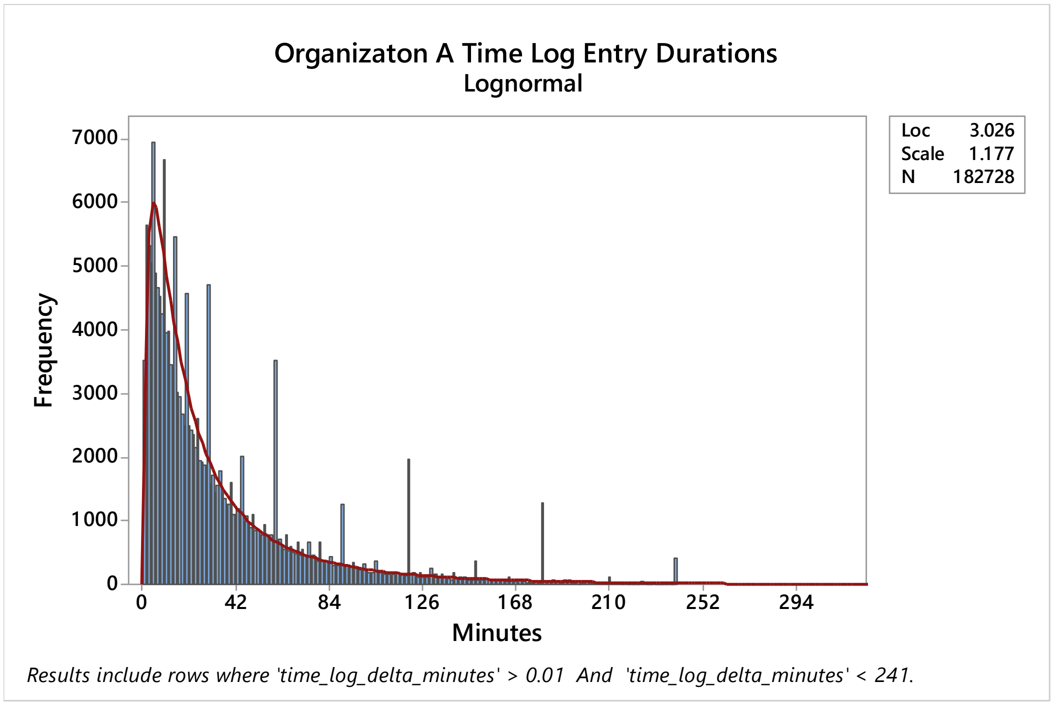


Figure 71: Time Log Entries (Organization A)

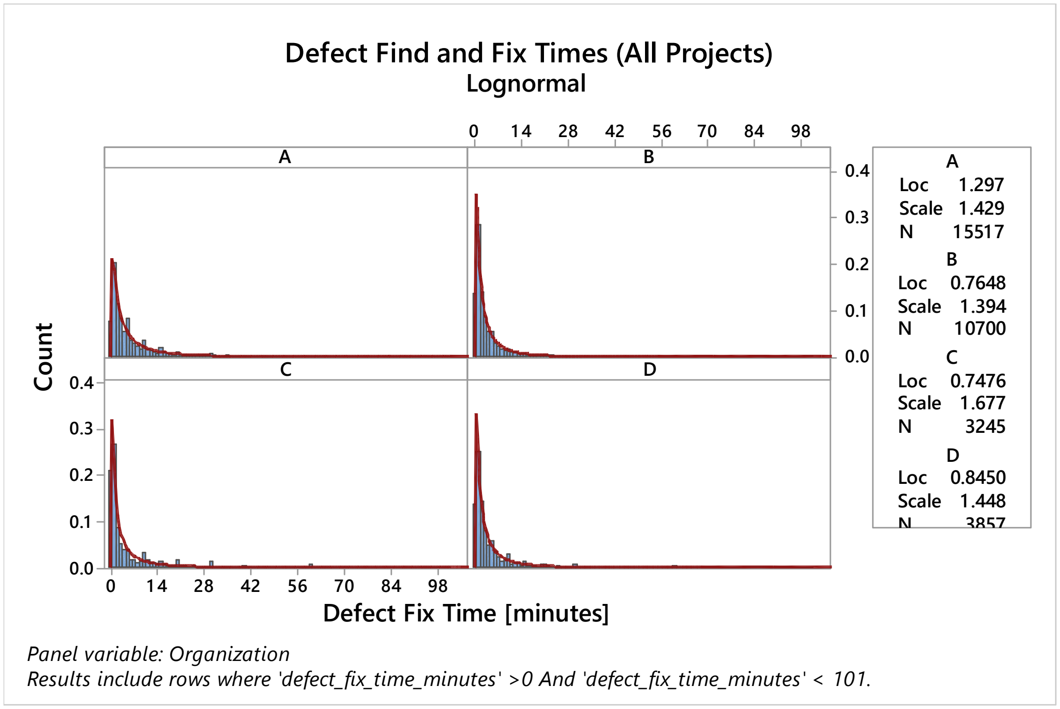


Figure 72: Defect Fix Time Distributions for Four Organizations

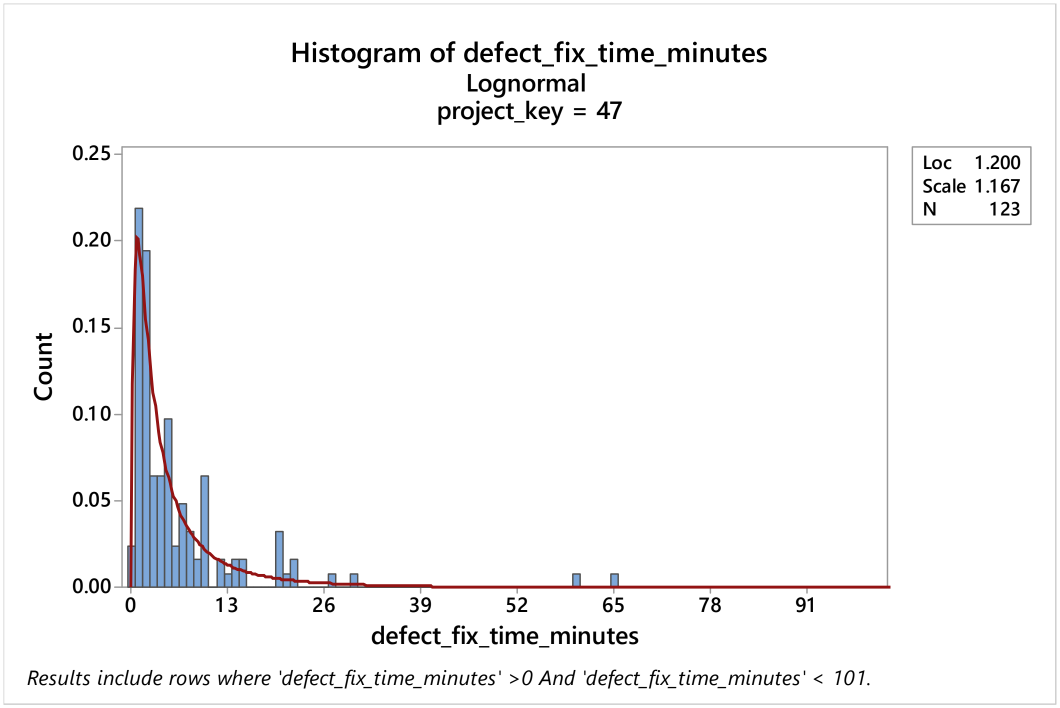


Figure 73: Defect Fix Time Distributions, Project 47

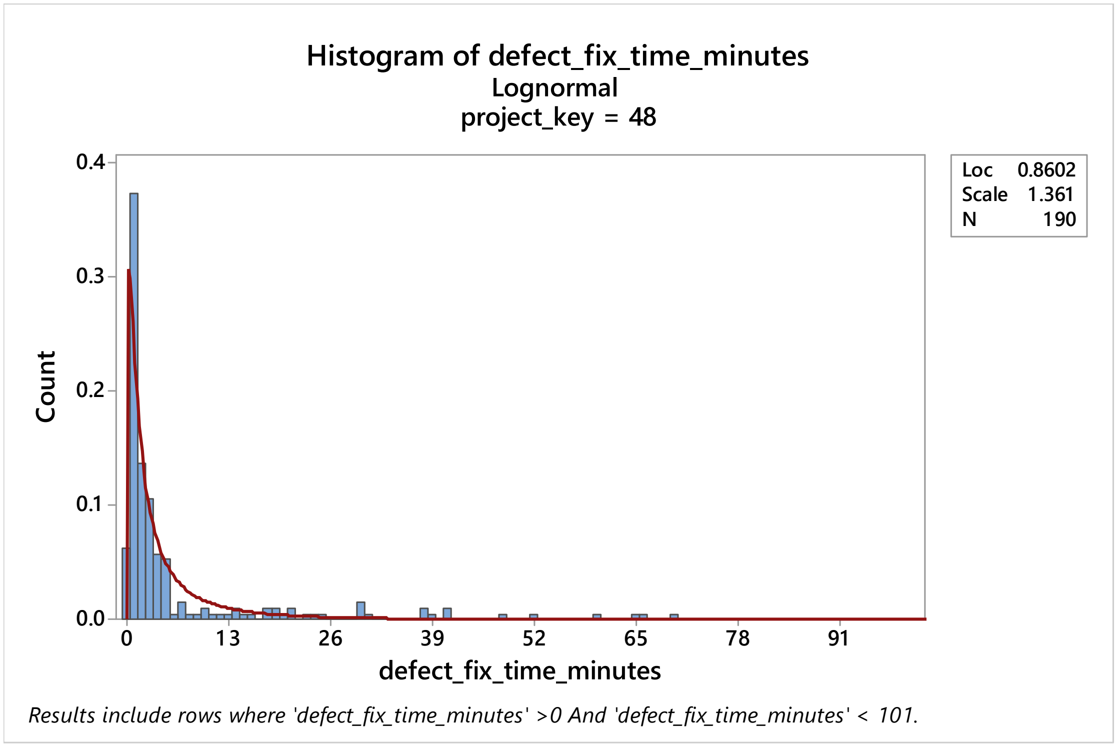


Figure 74: Defect Fix Time Distributions, Project 48

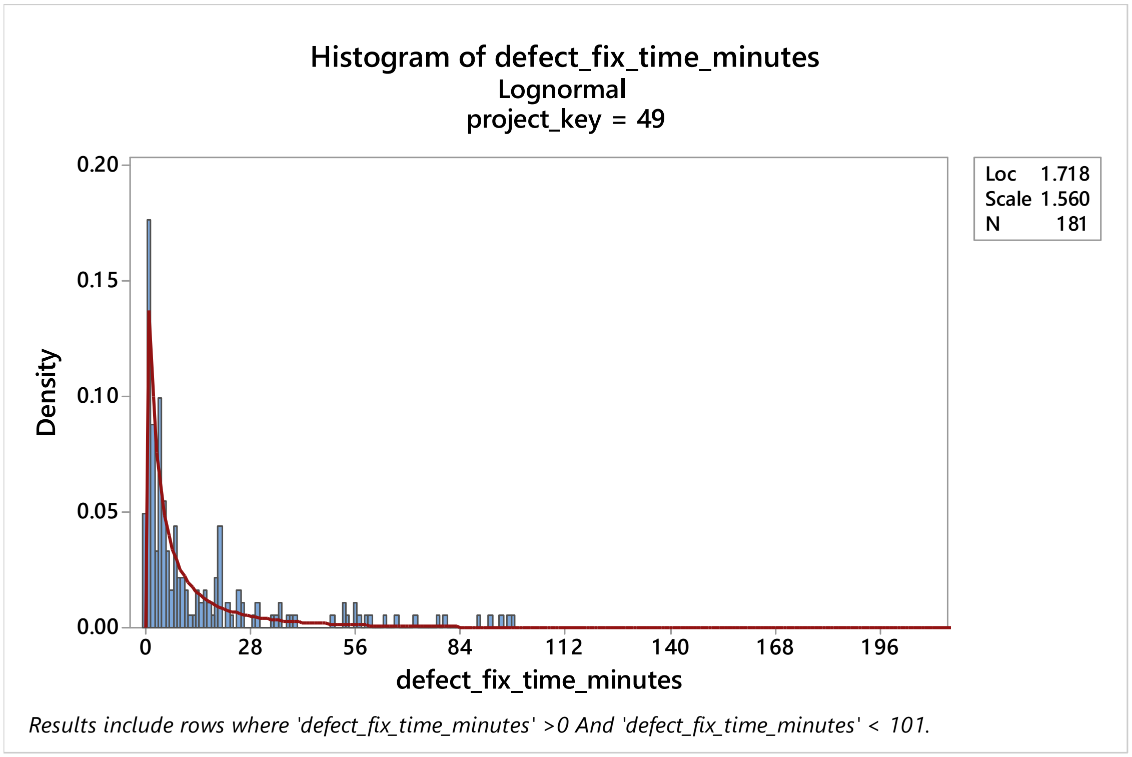


Figure 75: Defect Fix Time Distributions, Project 49

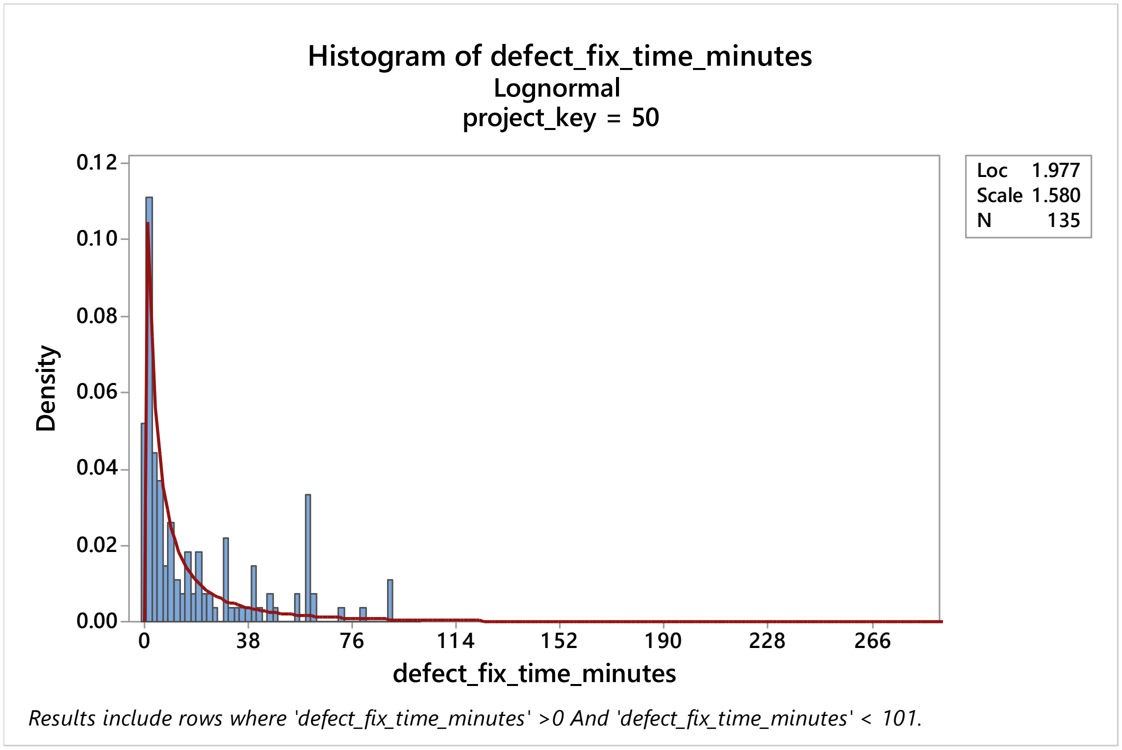


Figure 76: Defect Fix Time Distributions, Project 50

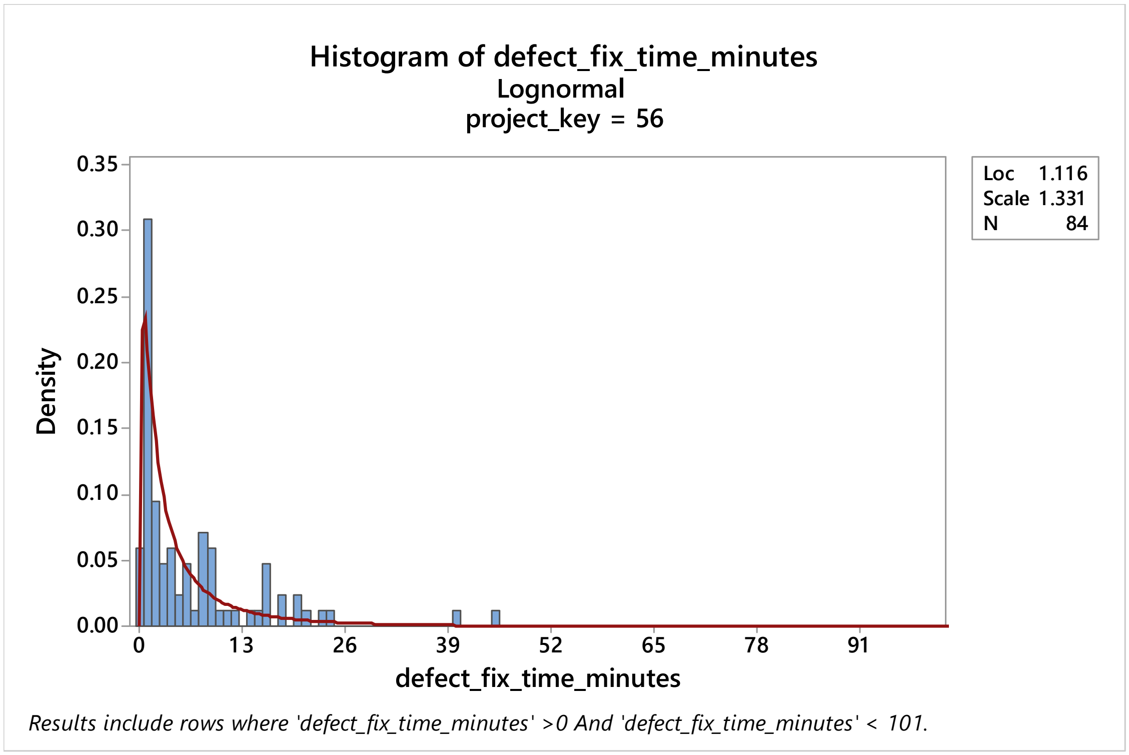


Figure 77: Defect Fix Time Distributions, Project 56

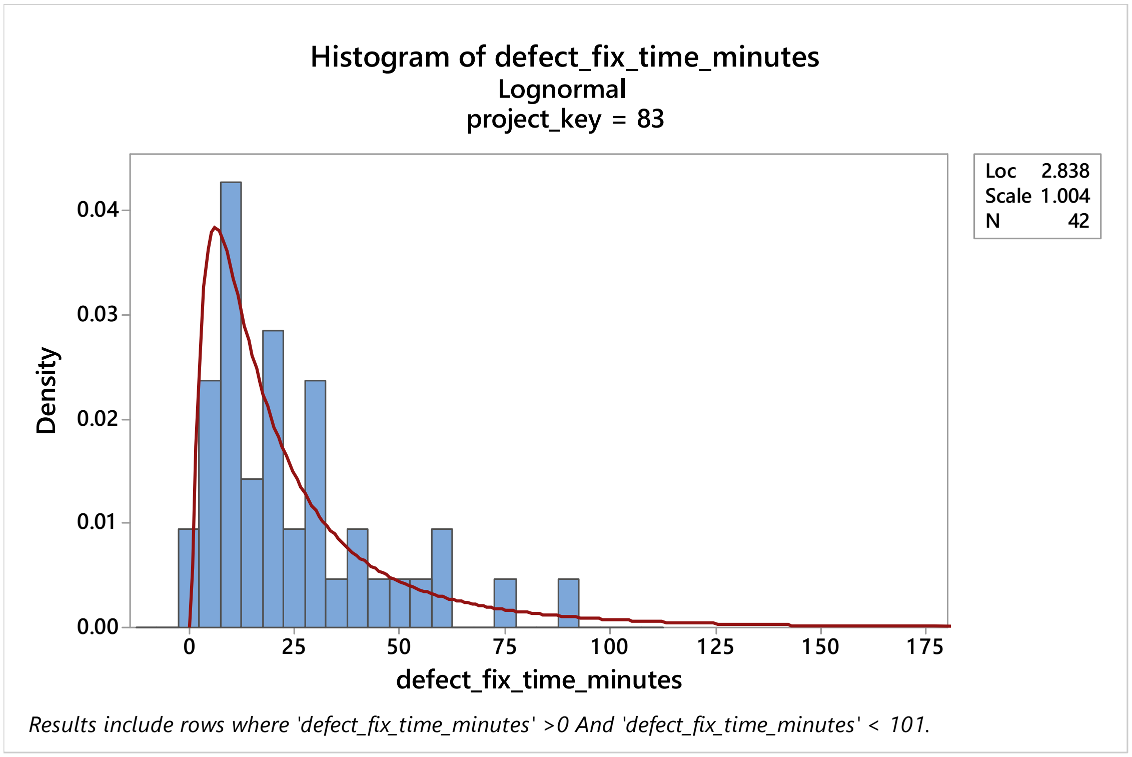


Figure 78: Defect Fix Time Distributions, Project 83

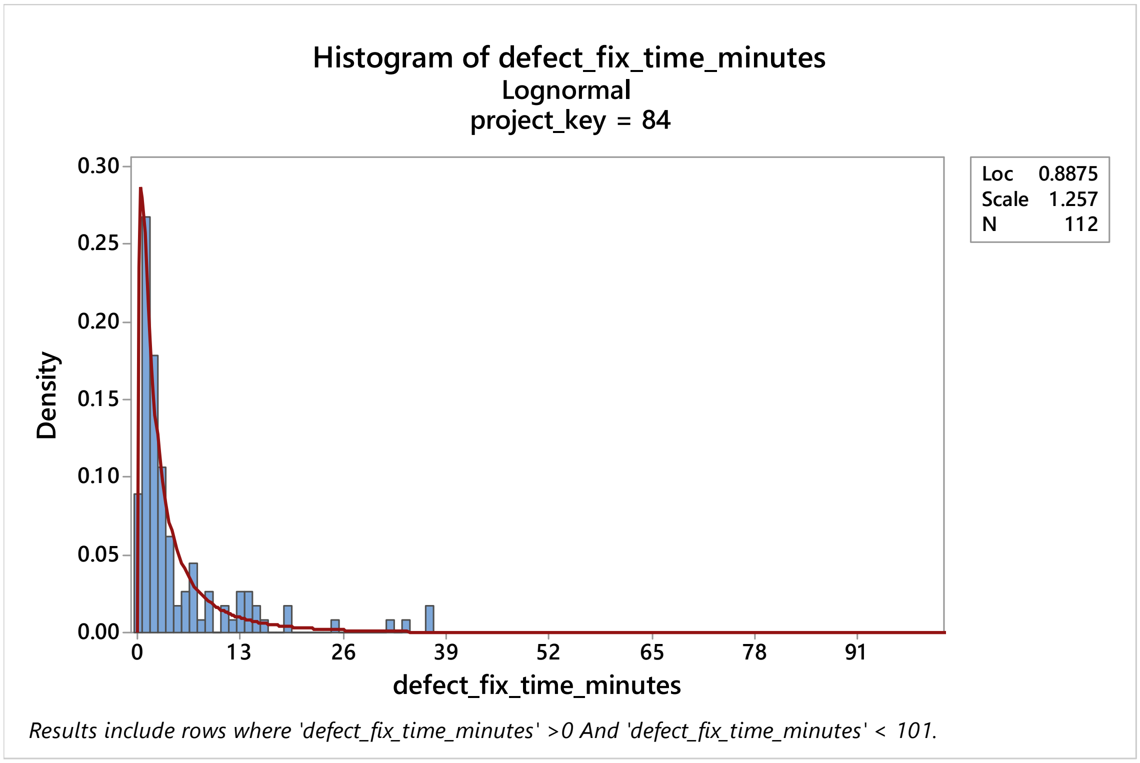


Figure 79: Defect Fix Time Distributions, Project 84

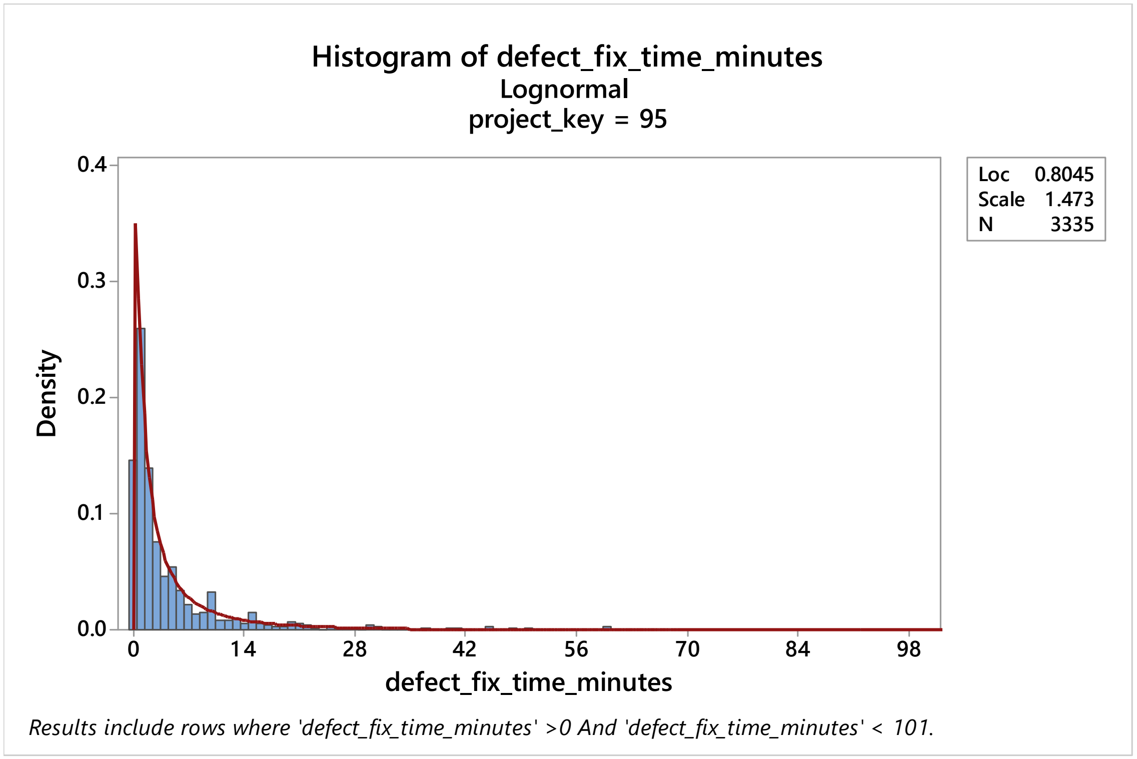


Figure 80: Defect Fix Time Distributions, Project 95

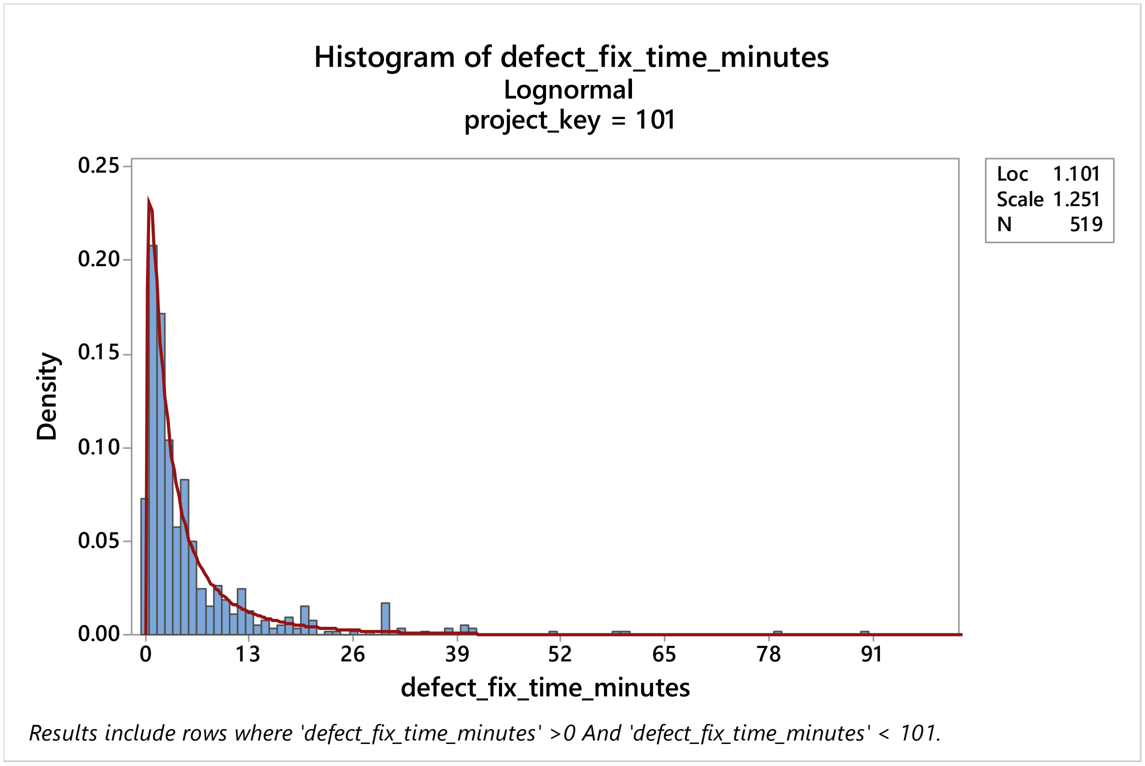


Figure 81: Defect Fix Time Distributions, Project 101

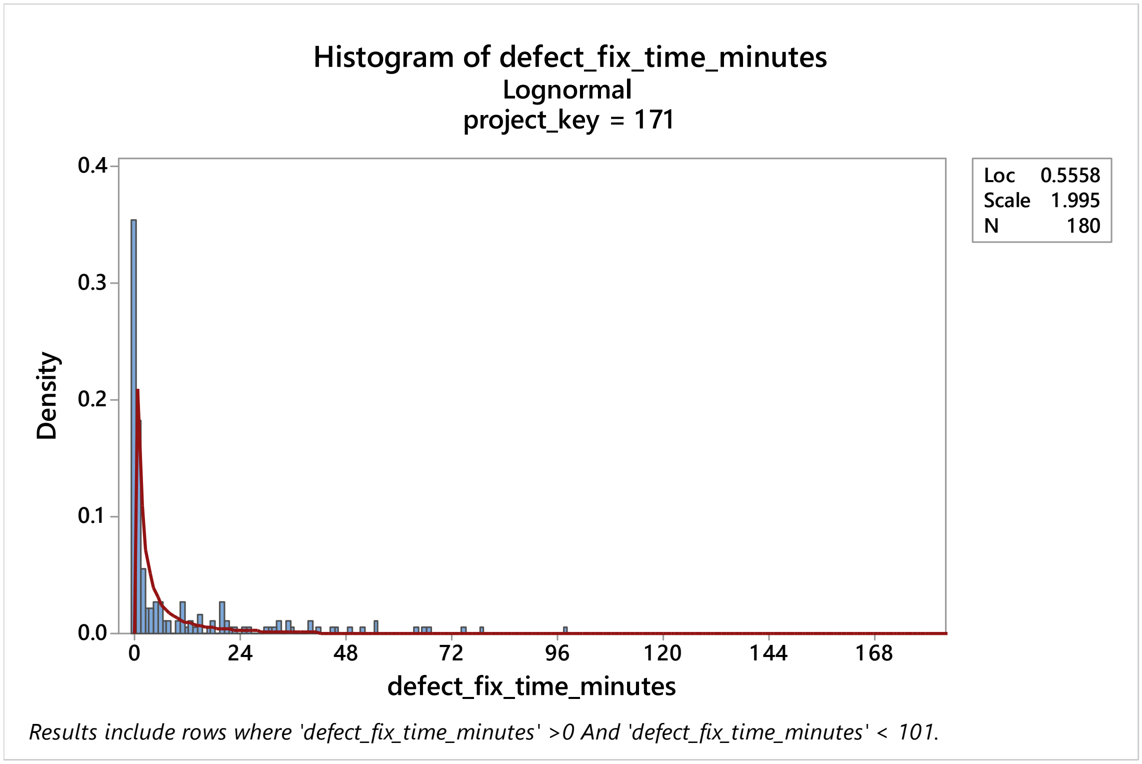


Figure 82: Defect Fix Time Distributions, Project 171

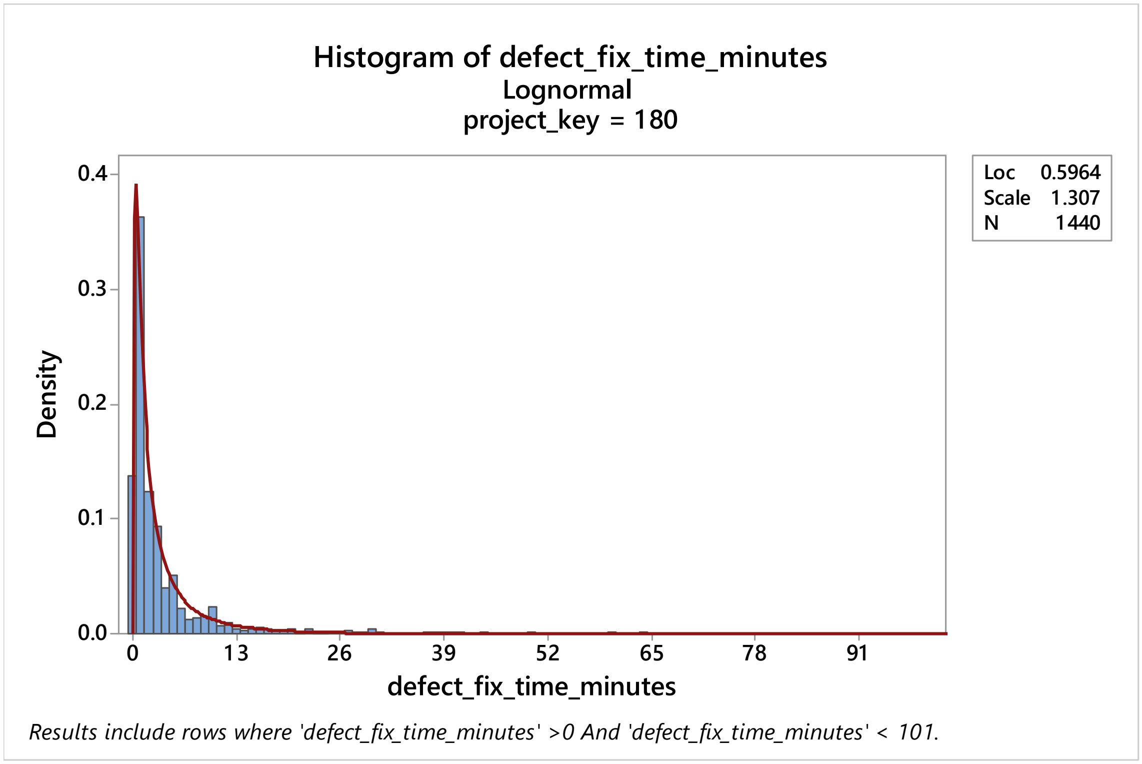


Figure 83: Defect Fix Time Distributions, Project 180

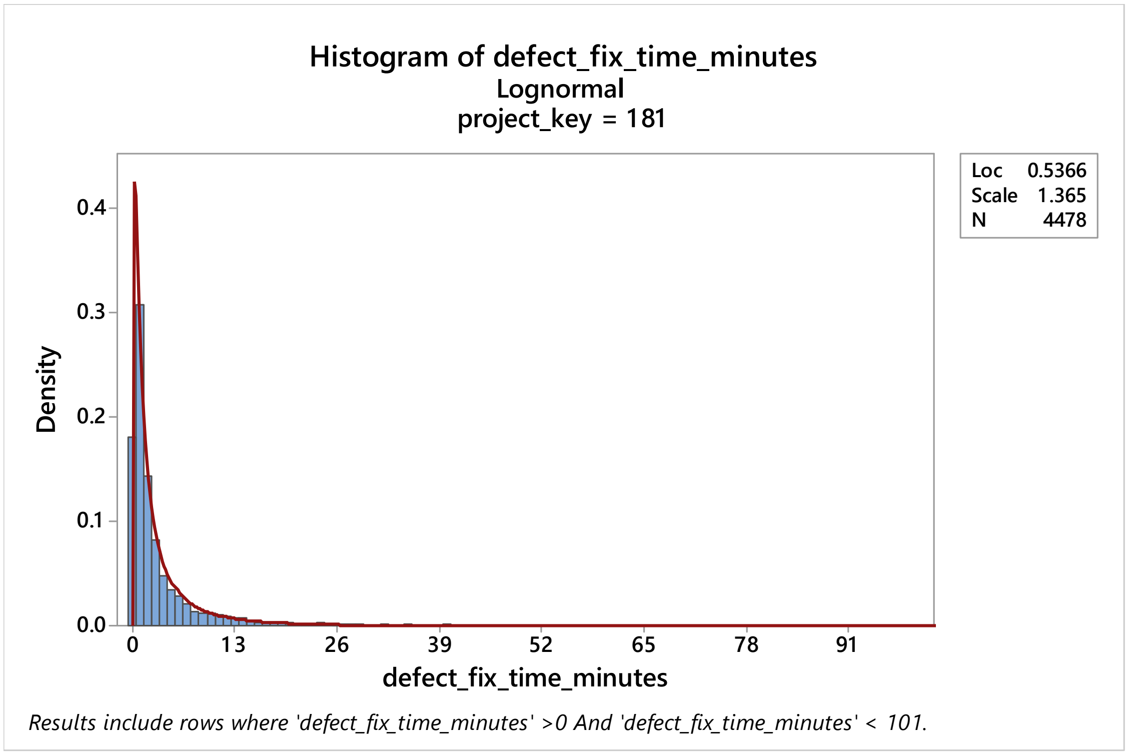


Figure 84: Defect Fix Time Distributions, Project 181

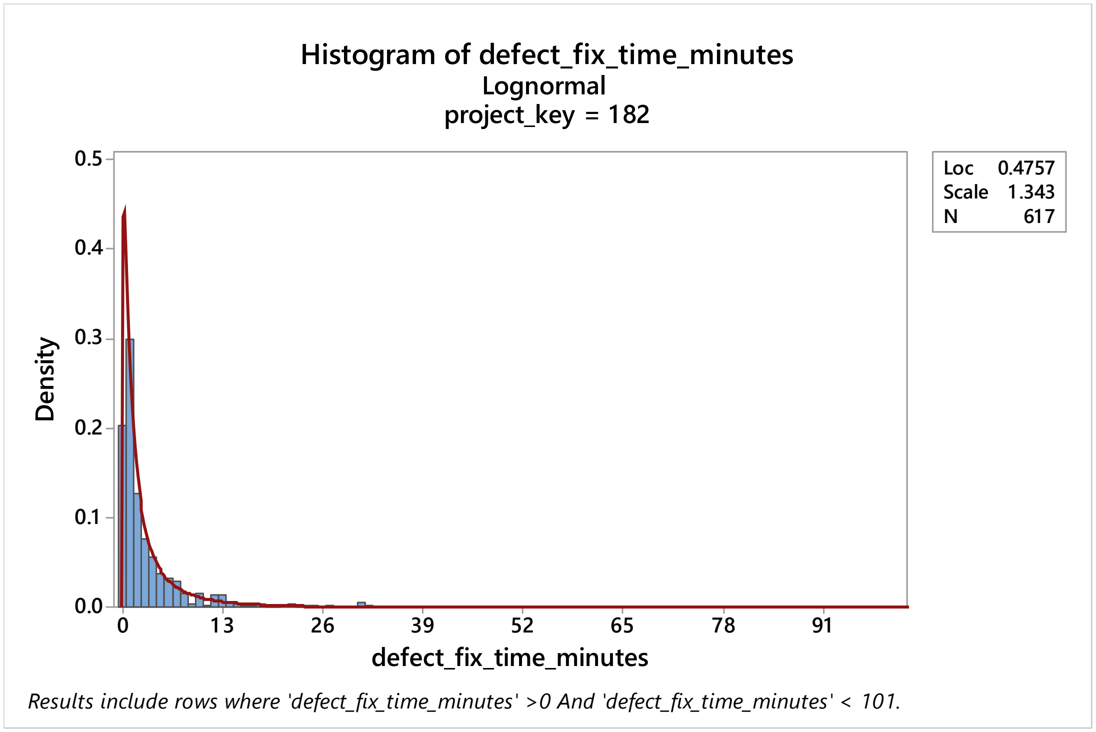


Figure 85: Defect Fix Time Distributions, Project 182

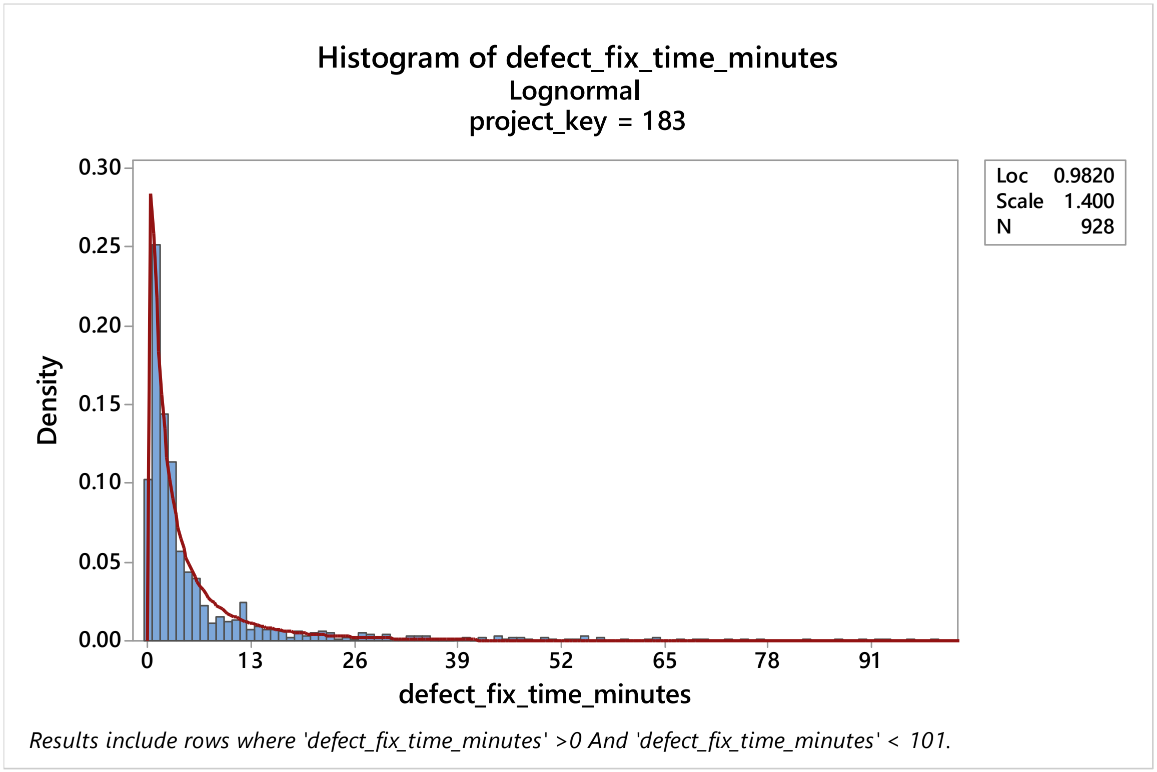


Figure 86: Defect Fix Time Distributions, Project 183

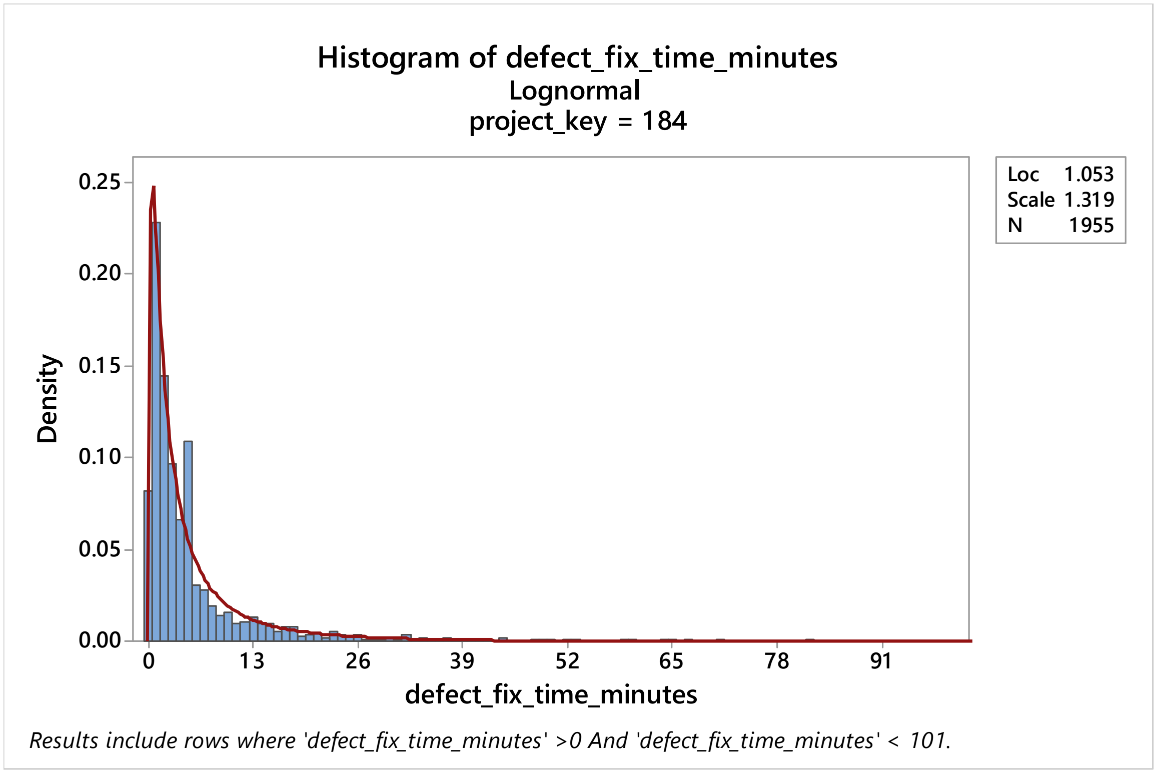


Figure 87: Defect Fix Time Distributions, Project 184

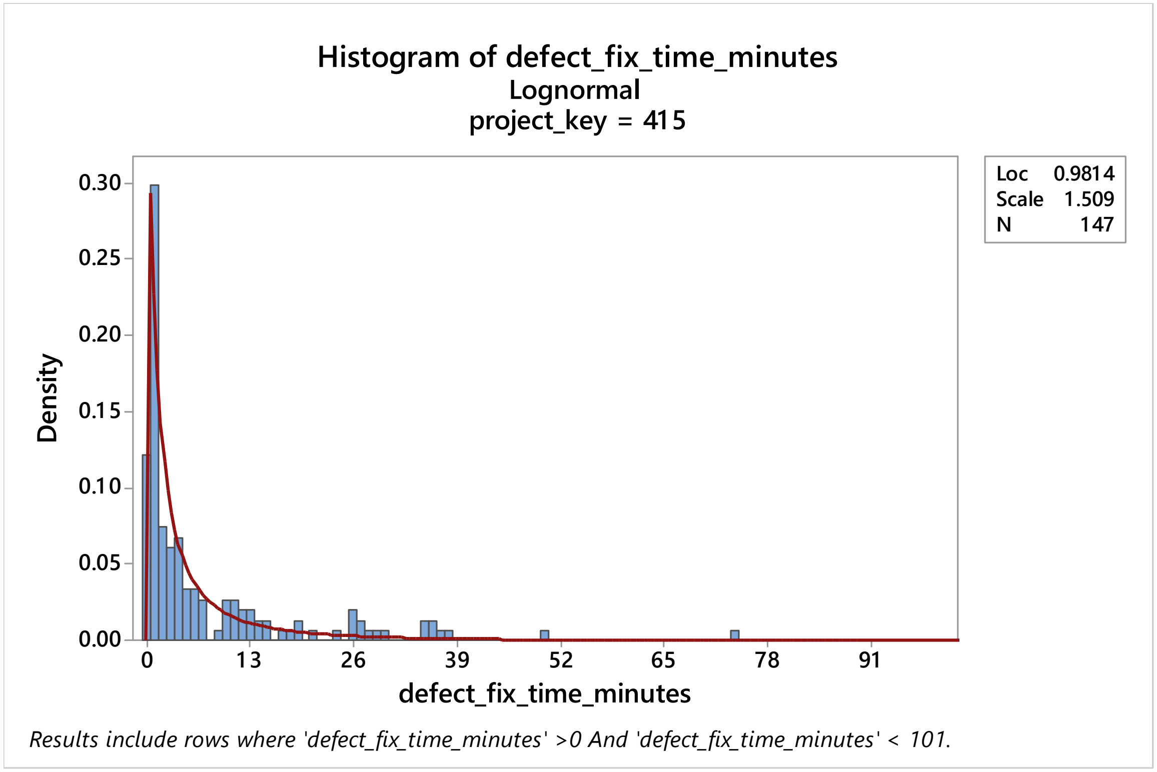


Figure 88: Defect Fix Time Distributions, Project 415

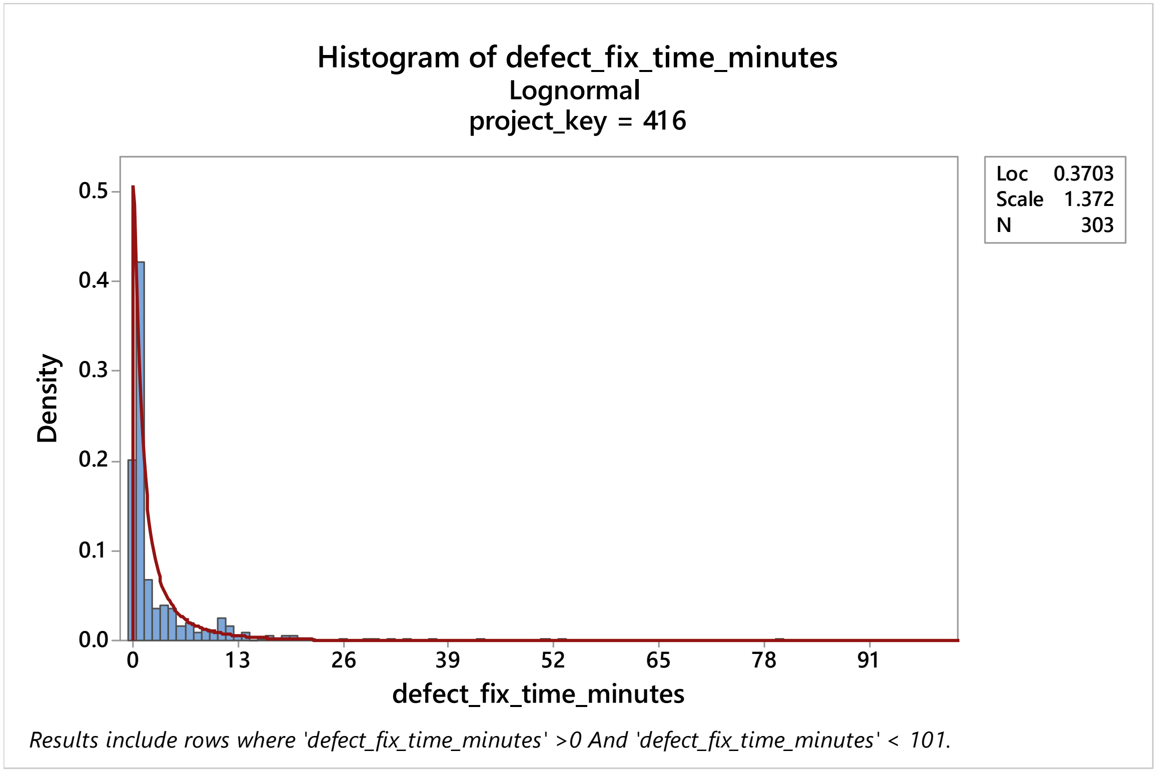


Figure 89: Defect Fix Time Distributions, Project 416

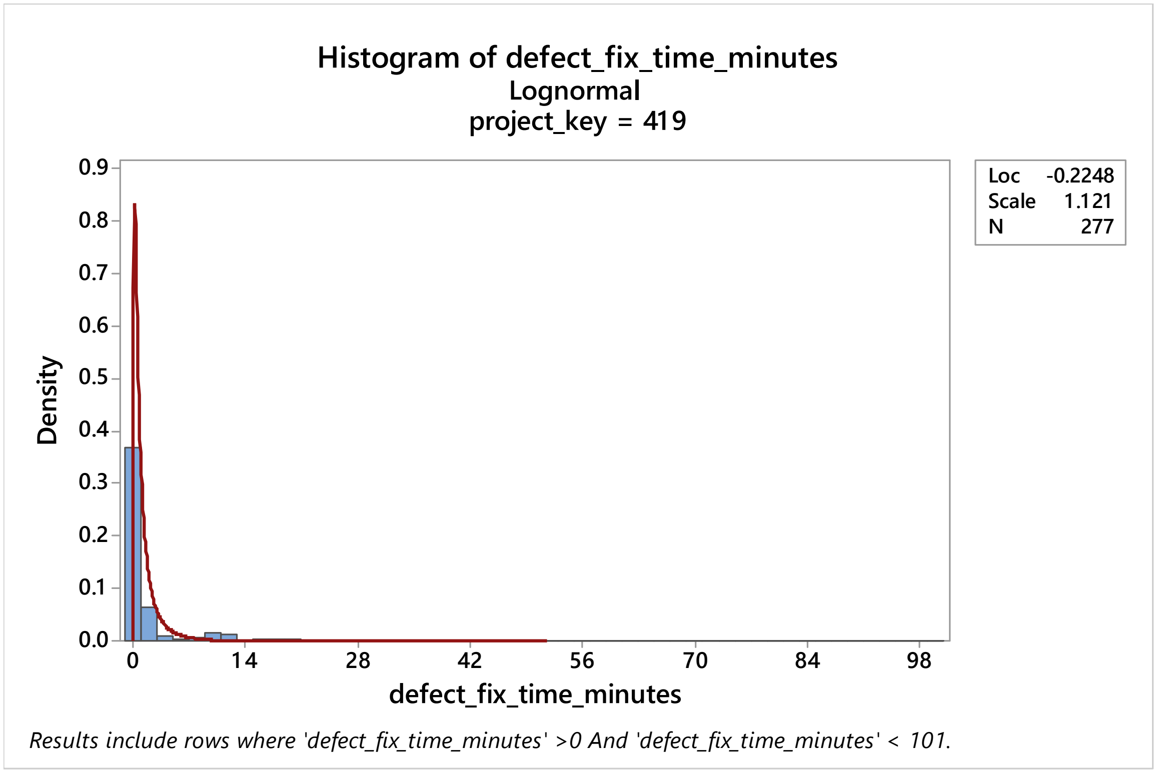


Figure 90: Defect Fix Time Distributions, Project 419

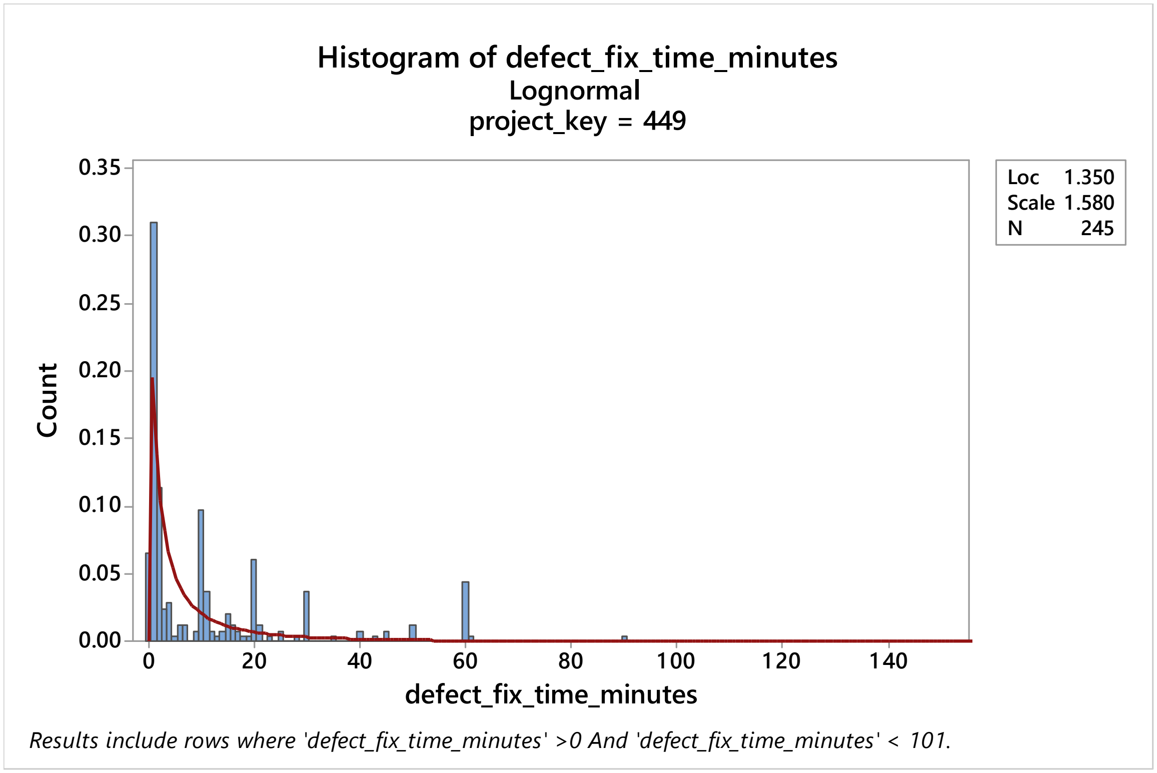


Figure 91: Defect Fix Time Distributions, Project 449

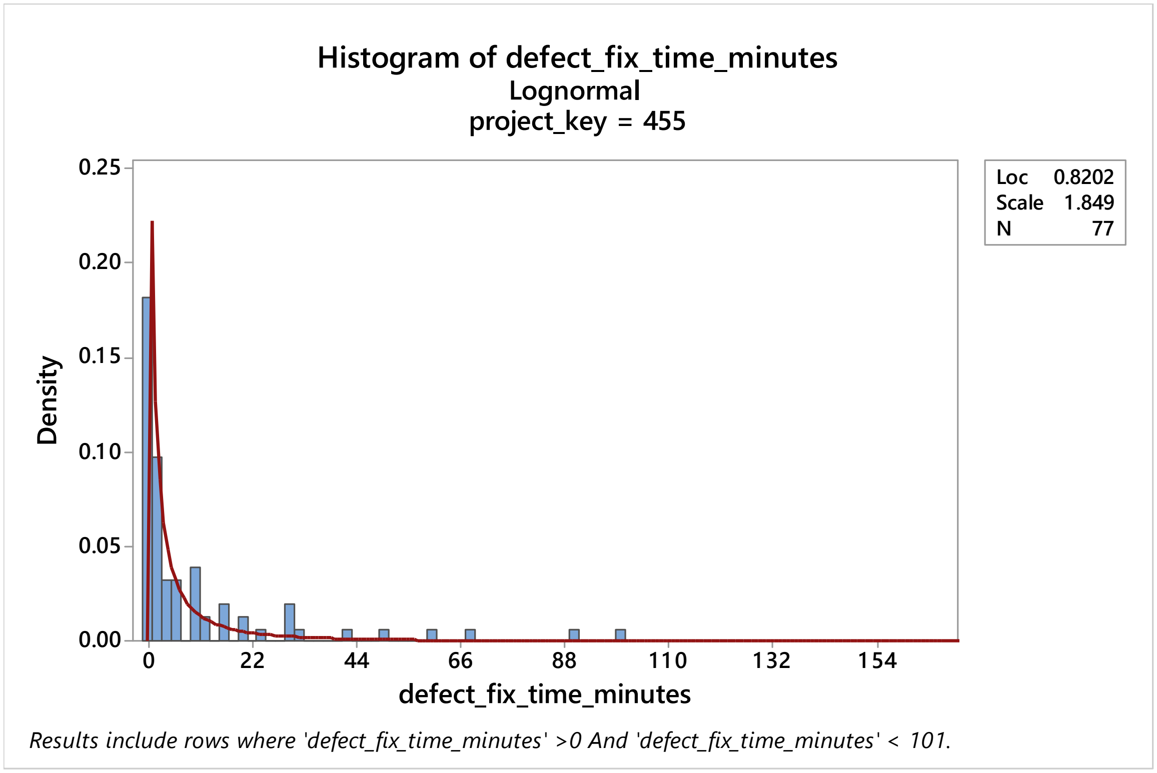


Figure 92: Defect Fix Time Distributions, Project 455

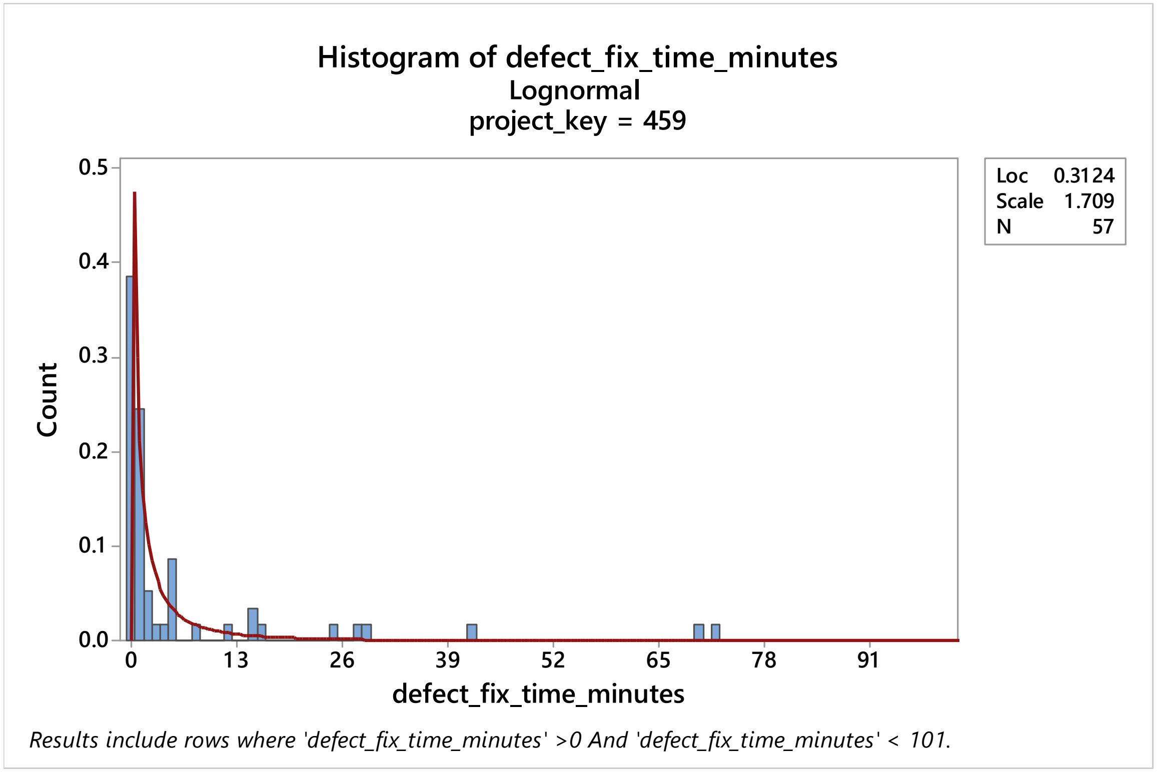


Figure 93: Defect Fix Time Distributions, Project 459

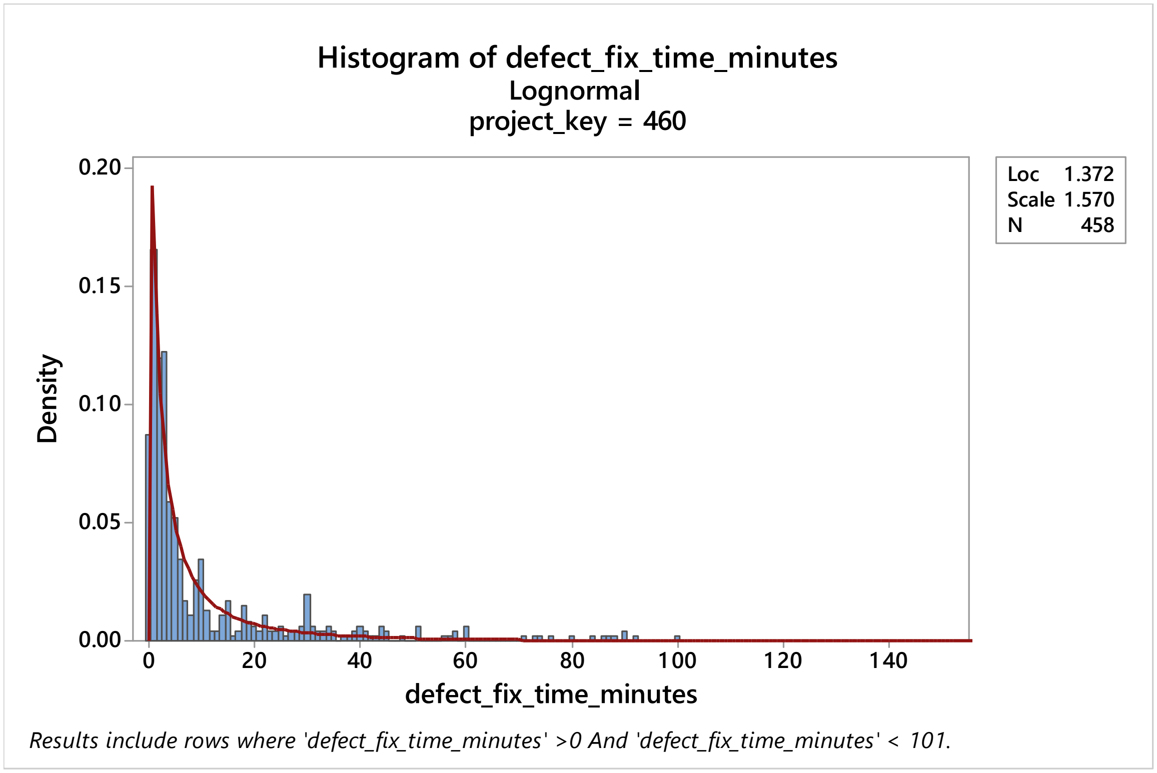


Figure 94: Defect Fix Time Distributions, Project 460

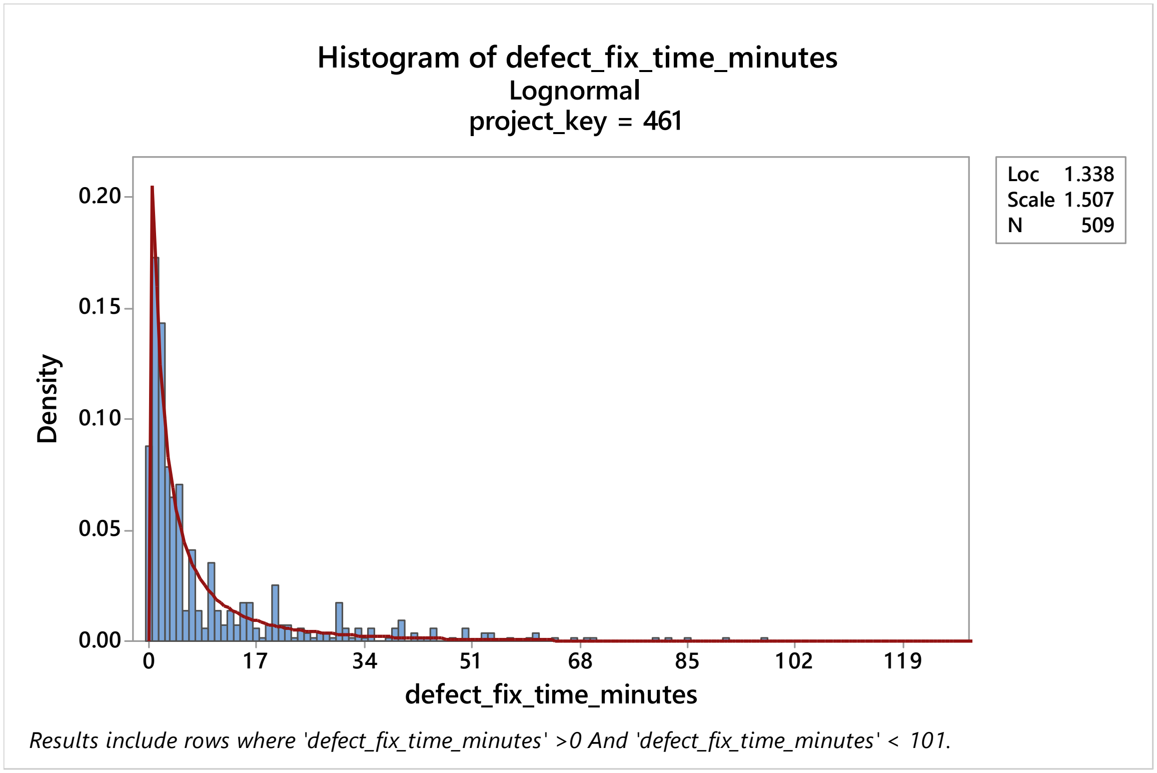


Figure 95: Defect Fix Time Distributions, Project 461

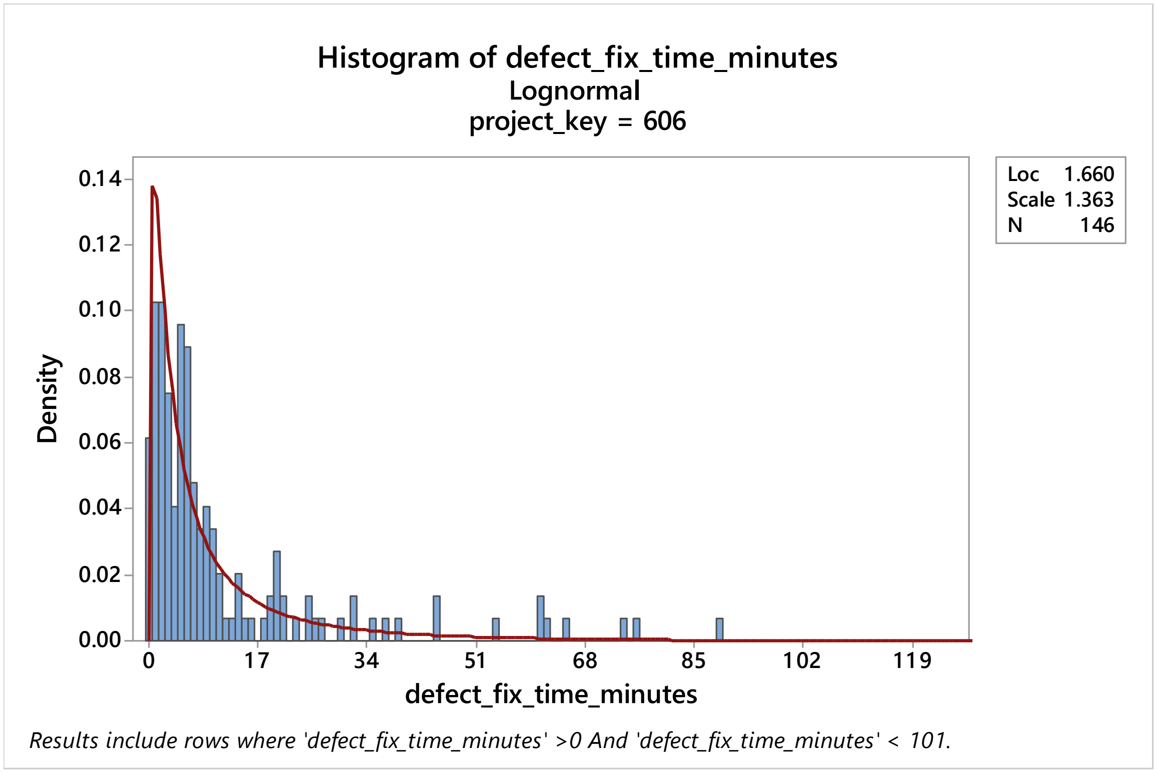


Figure 96: Defect Fix Time Distributions, Project 606

References/Bibliography

[Baldwin 2011]
Baldwin, Kristen; Dahmann, Judith; & Goodnight, Jonathan. Systems of Systems and Security: A Defense Perspective. *INCOSE Insight*. Volume 14. Issue 2. July 2011. Pages 11-13.

[Bartol 2008]
Bartol, Nadya. *Practical Measurement Framework for Software Assurance and Information Security, Version 1.0*. Practical Software and Systems Measurement. October 1, 2008. http://www.psmsc.com/Downloads/TechnologyPapers/SwA%20Measurement%2010-08-08.pdf

[Caivano 2005]
Caivano, Danilo. Continuous Software Process Improvement through Statistical Process Control. Pages 288-293. *9th European Conference on Software Maintenance and Reengineering*. IEEE Press, Manchester. March 2005. https://doi.org/10.1109/CSMR.2005.20

[Caralli 2010]
Caralli, Richard; Allen, Julia; Curtis, Pamela; White, David; & Young, Lisa. *CERT Resilience Management Model, Version 1.0.* CMU/SEI-2010-TR-012. Software Engineering Institute, Carnegie Mellon University. 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9479

[Chulani 1999]
Chulani, Sunita & Boehm, Barry. *Modeling Software Defect Introduction and Removal: COQUALMO (COnstructive QUALity MOdel).* 1999. http://sunset.usc.edu/TECHRPTS/1999/usccse99-510/usccse99-510.pdf

[DoD 2017]
Office of Small Business Programs, Department of Defense. Cybersecurity Challenges Protecting DoD's Unclassified Information. 2017. http://business.defense.gov/Portals/57/Documents/Public%20Meeting%20-%20Jun%2023%202017%20Final.pdf?ver=2017-06-26-143959-047

[Dybå 2012]
Dybå, Tore; Sjøberg, Dag; & Cruzes, Daniela. What Works for Whom, Where, When, and Why? On the Role of Context in Empirical Software Engineering. 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). September 2012. https://ieeexplore.ieee.org/document/6475393/

[Dybå 2013]
Dybå, Tore. Contextualizing Empirical Evidence. *IEEE Software*. Volume 30. Issue 1. January-February 2013. Pages 81-83.

[Ebert 1998]
Ebert, Christof. The Quest for Technical Controlling. *Software Process: Improvement and Practice*. Volume 4. Issue 1. March 1998. Pages 21-31. https://doi.org/10.1002/(SICI)1099-1670(199803)4:1<21::AID-SPIP92>3.0.CO;2-Q

[Emanuelsson 2008]
Emanuelsson, Pär & Nilsson, Ulf. A Comparative Study of Industrial Static Analysis Tools. *Electronic Notes in Theoretical Computer Science*. Volume 217. July 21, 2008. Pages 5–21. https://doi.org/10.1016/J.ENTCS.2008.06.039

[Forrester 2006]
Forrester, Eileen. *A Process Research Framework.* Software Engineering Institute, Carnegie Mellon University. December 2006. ISBN: 0-9786956-1-5. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=30501

[Fritz 2003]
Fritz, Julie & Cleland, Joshua. Effectiveness versus Efficacy: More Than a Debate Over Language. *Journal of Orthopaedic & Sports Physical Therapy*. Volume 33. Issue 4. 2003. Pages 163–165. https://doi.org/10.2519/jospt.2003.33.4.163

[Gartlehner 2006]
Gartlehner, Gerald; Hansen, Richard; Nissman, Daniel; Lohr, Kathleen; & Carey, Timothy. Criteria for Distinguishing Effectiveness from Efficacy Trials in Systematic Reviews. *Technical Review 12* (Prepared by the RTI-International-University of North Carolina Evidence-based Practice Center under Contract No. 290-02-0016.) AHRQ Publication No. 06-0046. Rockville, MD: Agency for Healthcare Research and Quality. April 2006.

[Gibson 2006]
Gibson, Diane; Goldenson, Dennis; & Kost, Keith. *Performance Results of CMMI-Based Process Improvement.* CMU/SEI-2006-TR-004. Software Engineering Institute, Carnegie Mellon University. 2006. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8065

[Hare 1995]
Hare, L.B., Hoerl, R.W., Hromi, J.D., and Snee, R.D. The Role of Statistical Thinking in Management. *Quality Progress*. February 1995. Pages 53-60.

[Heffley 2004]
Heffley, Jon & Meunier, Pascal. Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security? *Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004*. https://doi.org/10.1109/HICSS.2004.1265654

[Henry 1995]
Henry, Joel; Rossman, Allan; & Snyder, John. Quantitative Evaluation of Software Process Improvement. *Journal of Systems and Software*, Volume 28, Issue 2, Pages 169–177. https://doi.org/10.1016/0164-1212(94)00053-P

[Howard 2007]
Howard, Michael & Lipner, Steve. *The Security Development Lifecycle*. Microsoft Press. May 2006. http://download.microsoft.com/download/f/c/7/fc7d048b-b7a5-4add-be2c-baaee38091e3/9780735622142_SecurityDevLifecycle_ch01.pdf

[Humphrey 1999]
Humphrey, Watts. *Introduction to the Team Software Process*. Addison-Wesley Professional. 1999.

[Humphrey 2010]
Humphrey, Watts; Chick, Timothy; Nichols, William; & Pomeroy-Huff, Marsha. *Team Software Process (TSP) Body of Knowledge (BOK)*. CMU/SEI-2010-TR-020. Software Engineering Institute, Carnegie Mellon University. 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9551

[Jones 2011]
Jones, Capers & Bonsignour, Olivier. *The Economics of Software Quality*. Addison-Wesley Professional. 2011. ISBN-13: 978-0132582209

[Kaner 2004]
Kaner, Cem & Bond, Walter. *Software Engineering Metrics: What Do They Measure and How Do We Know?* 10th International Software Metrics Symposium, Metrics. 2004. http://www.kaner.com/pdfs/metrics2004.pdf

[Kitchenham 1996]
Kitchenham, B & Pfleeger, S. Software Quality: The Elusive Target. *IEEE Software*. Volume 13. Issue 1. January 1996. Pages 12–21. https://doi.org/10.1109/52.476281

[Madachy 2008]
Madachy, Raymond & Boehm, Barry. Assessing Quality Processes with ODC COQUALMO. In: Wang Q., Pfahl D., Raffo D.M. (eds) Making Globally Distributed Software Development a Success Story. ICSP 2008. *Lecture Notes in Computer Science*. Volume 5007. Springer.

[Martin 2014]
Martin, Robert. Non-Malicious Taint: Bad Hygiene Is as Dangerous to the Mission as Malicious Intent. *CrossTalk*. Volume 2. March/April 2014. Pages 4–9.

[Mead 2010]
Mead, Nancy & Allen, Julia. *Building Assured Systems Framework*. CMU/SEI-2010-TR-025. Software Engineering Institute, Carnegie Mellon University. 2010. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9611

[Nichols 2012]
Nichols, William. Plan for Success, Model the Cost of Quality. *Software Quality Professional*. Volume 14. Issue 2. March 2012. Pages 4–11.

[Paulish 1993]
Paulish, Daniel. *Case Studies of Software Process Improvement Methods*. CMU/SEI-93-TR-026. Software Engineering Institute, Carnegie Mellon University. 1993. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11977

[Paulish 1994]
Paulish, Daniel & Carleton, Anita. Case Studies of Software-Process-Improvement Measurement. *Computer*. Volume 27. Issue 9. 1994. Pages 50–57. https://doi.org/10.1109/2.312039

[Paulk 2009]
Paulk, M.; Needy, L.; & Rajgopal, J. Identify Outliers, Understand the Process. *ASQ Software Quality Professional*, Volume 11. Issue 2. March 2009. Pages 28-37.

[Peltzman 1975]
Peltzman, Sam. The Effects of Automobile Safety Regulation. *Journal of Political Economy*. Volume 83. Issue 4. August 1975. Pages 677-726. https://doi.org/10.1086/260352

[Petersen 2009]
Petersen, Kai & Wohlin, Claes. Context in Industrial Software Engineering Research. *ESEM '09 Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. Pages 401–404. https://doi.org/10.1109/ESEM.2009.5316010

[Rozum 1993]
Rozum, James. *Concepts on Measuring the Benefits of Software Process Improvement.* CMU/SEI-93-TR-009. Software Engineering Institute, Carnegie Mellon University. 1993. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11871

[Runeson 2008]
Runeson, P. & Höst, M. Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*. Volume 14. 2009. Page 131–164. https://doi.org/10.1007/s10664-008-9102-8

[Runeson 2012]
Runeson, Per; Host, Martin; Rainer, Austen; & Regnell, Bjorn. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons Inc. April 2012.

[Salazar 2014]
Salazar, Rafael; Mejorado, Antonio; & Nichols, William. TSP-PACE: Process and Capability Evaluation, an Experience Report. *TSP Symposium 2014 Proceedings and Presentations*. Pages 378–383. Software Engineering Institute, Carnegie Mellon University.

[Schneidewind 1999]
Schneidewind, Norman. Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics. *IEEE Transactions on Software Engineering*. Volume 25. Issue 6. November/December 1999. Pages 769 – 781. https://doi.org/10.1109/ICSM.1997.624250

[Shin 2011]
Shin, Yonghee & Williams, Laurie. Can Traditional Fault Prediction Models Be Used for Vulnerability Prediction? *Empirical Software Engineering*. Volume 18. 2011. Pages 25–59. https://doi.org/10.1007/s10664-011-9190-8

[Shirai 2015]
Shirai Yasutaka & Nichols, William. *Project Fact Sheets from the Team Software Process SEMPR Warehouse*. CMU/SEI-SR-007. 2015. Software Engineering Institute, Carnegie Mellon University.

[Shirai 2014]
Shirai, Yasutaka; Nichols, William; & Kasunic, Mark. Initial Evaluation of Data Quality in a TSP Software Engineering Project Data Repository. *ICSSP 2014 Proceedings of the 2014 International Conference on Software and System Process*. Pages 25–29. ACM. https://doi.org/10.1145/2600821.2600841

[Singal 2014]
Singal, A.; Higgins, P.; & Walijee, A. A Primer on Effectiveness and Efficacy Trials. *Clinical and Translational Gastroenterology*. Volume 5. Issue 2. January 2014. https://doi.org/10.1038/ctg.2013.13

[Snyder, 2015]
Snyder, Don; Powers, James; Bodine-Baron, Elizabeth; Fox, Bernard; Kendrick, Lauren; & Powell, Michael. *Improving the Cybersecurity of U.S. Air Force Military Systems Throughout Their Lifecycles.* Rand Research Report. 2015. ISBN: 978-0-8330-8900-7. https://www.rand.org/content/dam/rand/pubs/research_reports/RR1000/RR1007/RAND_RR1007.pdf

[SPDI 2014]
Software Process Dashboard Initiative. TSP Process Dashboard Data Warehouse. 2014. http://www.processdash.com/tpdw

[SPDI 2017]
Software Process Dashboard Initiative. The Software Process Dashboard. August 5, 2017. http://www.processdash.com/

[Unterkalmsteiner 2012]
Unterkalmsteiner, Michael; Gorschek, Tony; Islam, Moinul; Cheng, Chow Kian, Permadi, Rahadian Bayu; and Feldt, Robert. Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review. *IEEE Transactions on Software Engineering.* Volume 38. Issue 2. Pages 398-424. DOI: 10.1109/TSE.2011.26

[Vallespir 2011]
Vallespir, Diego & Nichols, William. Analysis of Design Defect Injection and Removal in PSP. TSP Symposium 2011. Pages 1–28. Software Engineering Institute, Carnegie Mellon University.

[Vallespir 2012]
Vallespir, Diego & Nichols, William. An Analysis of Code Defect Injection and Removal in PSP. *Proceedings of the TSP Symposium 2012*. Software Engineering Institute, Carnegie Mellon University.

[van Solingen, 2004]
van Solingen, Rini. Measuring the ROI of Software Process Improvement. *IEEE Software.* Volume 21. Issue 3. May-June 2004. Pages 32-38. https://doi.org/10.1109/MS.2004.1293070

[Westfall 2007]
Westfall, John; Mold, James; & Fagnan, Lyle. Practice-Based Research—“Blue Highways” on the NIH Roadmap. *JAMA*. Volume 297. Issue 4. 2007. Pages 403–406. doi:10.1001/jama.297.4.403

[Wheeler 2016]
Wheeler, David & Henninger, Amy. *State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation V2.2*. Institute for Defense Analysis. 2016. https://doi.org/IDA Paper P-5061

[Woody 2015]
Woody, C.; Ellison, R.; & Nichols, W. Predicting Cybersecurity Using Quality Data. 2015 IEEE International Symposium on Technologies for Homeland Security. 2015. Pages 1–5. https://doi.org/10.1109/THS.2015.7225327

[Zheng 2006]
Zheng, Jiang; Williams, Laurie; Nagappan, Nachiappan; Snipes, Will; Hudepohl, John; & Vouk, Mladen. On the Value of Static Analysis for Fault Detection in Software. *IEEE Transactions on Software Engineering*. Volume 32. Issue 4. Pages 240–253. https://doi.org/10.1109/TSE.2006.38

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Publicreportingburdenforthiscollectionofinformationisestimat-edtoaverage1hourperresponse,includingthetimeforreviewinginstructions,searchingexistingdatasources,gatheringandmaintainingthedataneeded,andcom-pletingandreviewingthecollectionofinformation.Sendcommentsregardingthisburdenestimateoranyotheraspectofthis-collectionofinformation,includingsuggestionsforreducingthisburden,toWashingtonHeadquartersServices,DirectorateforinformationOperationsandReports,1215JeffersonDavisHighway,Suite1204,Arlington,VA22202-4302,andtotheOfficeofManagementandBudget,PaperworkReductionProject(0704-0188),Washington,DC20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE October 2018		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Composing Effective Software Security Assurance Workflows			5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) William R. Nichols, James D. McHale, David Sweeney, William Snavely, & Aaron Volkman			
7. PERFORMING ORGANIZATION NAME(S)AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2018-TR-004
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE
13. ABSTRACT(MAXIMUM 200 WORDS) In an effort to determine how to make secure software development more cost effective, the SEI conducted a research study to empirically measure the effects that security tools—primarily automated static analysis tools—had on costs (measured by developer effort and schedule) and benefits (measured by defect and vulnerability reduction). The data used for this research came from 35 projects in three organizations that used both the Team Software Process and at least one automated static analysis (ASA) tool on source code or source code and binary. In every case quality levels improved when the tools were used, though modestly. In two organizations, use of the tools reduced total development effort. Effort increased in the third organization, but defect removal costs were reduced compared to the costs of fixes in system test. This study indicates that organizations should employ ASA tools to improve quality and reduce effort. There is some evidence, however, that using the tools could "crowd out" other defect removal activities, reducing the potential benefit. To avoid overreliance, the tools should be employed after other activities where practicable. When system test cycles require expensive equipment, ASA tools should pre-cede test; otherwise, there are advantages to applying them after system test.			
14. SUBJECT TERMS Software measurement, software security, static analysis, automated static analysis, secure software development, case study, tools			15. NUMBER OF PAGES 125
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN7540-01-280-5500

Standard Form 298 (Rev.2-89) Prescribed by ANSI Std. Z39-18298-102