An Empirical Evaluation of System and Regression Testing

Mechelle Gittens
Hanan Lutfiyya
Michael Bauer
University of Western Ontario

David Godwin
Yong Woo Kim
Pramod Gupta
IBM

Abstract

We had the opportunity to conduct an empirical study in the context of the testing environment for a large commercial product. The particular goal of the organization for which this study was done, was to gain a strong understanding of how particular aspects of their testing practice impact on the quality of the released products. In this paper we present some of the results of that research as it relates to the verification of intuitive claims of those in this industrial environment, and documented claims from other research about the relationships between several parameters. The parameters of interest to the organization were: breadth of system and regression testing of software components defined by code coverage, number of defects discovered by an in-house test team prior to the release of those software components, and number of defects discovered by the customer in the field subsequent to the release of those software components.

1 Introduction

In the practice and research of software engineering, several claims and assumptions are made and held about software development practices and their impact on the quality of the product that is released. There is, however, a lack of quantitative evidence to support many of these claims, and much of the discussion in the area is anecdotal. There has been much discussion in [NF94] and [PPV00] about the fact that those in software development have not measured the efficiency or economy for most of the models that are proposed. More specifically, there are several claims pertaining to software quality and its relationship to testing, and the number of defects in software. Such assertions are relevant to those in industrial circumstances, and often guide their perception of the effectiveness of their efforts. This particularly so in the absence of in-house measurement programs to verify the productiveness of testing activities. The absence of such programs is stated by Carnegie Mellon University's Software Engineering Institute [TG96] which confirms that 75% of the world's software development organizations take an ad hoc, unmeasured approach to development. Industrially based empirical studies are therefore of utmost importance for the advancement of knowledge in disciplines such as software testing.

According to [PPV00] an empirical study is a test that compares what we believe to what we observe. It must therefore present a hypothesis and set out to prove or disprove it. We are therefore not discussing work where experiments are done and quantitative results are merely recorded. There must be some hypothetical assertion and evaluation of whether that assertion is met. The conclusions are not solely a product of what we observe, but are also influenced by this prior belief. The study is targeted at those beliefs and as such, will influence practice based on those ideas. We have noted some examples focussed on the issues of software measurement, testing, and the resulting quality. These include the work of Adams in [Ada84], Daskalantonakis in [Das92], Khoshgoftaar, Allen, Kalaichelvan, and Goel in [KAKG96], Fenton and Ohlsson in [FO00], Shen, Yu, Thebault, and Paulsen in [SYTP85], and Piwowarski, Ohba and Caruso in [POC93]. We discuss these in section two, however they hardly constitute a mature body of knowledge.

Our study is intended to add to work such as that of [Ada84], [Das92], [KAKG96], [FO00], [SYTP85], and [POC93]. We also add a new perspective on issues that currently offer intuitive and qualitative arguments but no quantitative empirical substance. We do not claim that this work is the final statement on these questions, but do believe that they contribute to a body of work that will be strengthened by the documentation of this research.

Our discussion proceeds as follows. In section two

we review other empirical studies that are relevant to testing and software quality, and that our study seeks to complement. We continue by explaining the terminology that is used in the paper in our third section. Section four introduces the hypotheses that this paper will prove or disprove, and then we present a brief description of the industrial product that we used to conduct our study in section five. We then explain the exact nature of the parameters that we have collected and give the reader an idea of how this data was collected. In section six we provide the reader with an understanding of how the hypotheses that we have stated are currently presented in the literature. This section includes a summary of how we analyzed the data and our own results, as well as an exploration of what the results have shown with respect to the documented hypotheses. Finally, the conclusions and our outlook to future analysis and studies are presented in section seven.

2 Related Work

In [Das92] the authors discuss a program of metrics collection and analysis instituted for Motorola. This initiative was undertaken, as is our current study, to better understand their development process and to increase productivity, quality, and development cycle time. The company implemented requirements that mandated that specific metrics be taken including measurements of delivered defects, delivered defects per size of product, customer reported errors, and software reliability. The stated goals enumerated these measurements and correlated directly with them. The researchers documented several specific questions that were to be answered, such as - *What is the currently known defect content of software delivered to customers, normalized by assembly equivalent source size?* They stated that they were able to answer these questions (even though the particular answers to questions such as those above are omitted, we believe due to their sensitive nature), and at the end of this study they saw measurable benefits to recording these metrics, which included the marked reduction of released software defect density to a mere 2% of what it was originally, over a period of three and a half years.

Another relevant empirical study is the work done in [FO00]. This research is actually the epitome of the kind of study that we document here. Fenton and Ohlsson outline hypotheses on the Pareto principle, which examines beliefs on the concentration of defects in code, the use of data on defects found in-house by the vendor to predict errors found by customers in the field, the prediction capability of size, and complexity metrics for defects, and the effects of test on quality. As the description of our own hypotheses indicates, our work is strongly related to this study. These investigators actually presented several counterintuitive discoveries, and confirmed a few widely held beliefs. For example, they found weak support for the hypothesis that a higher number of defects found in function level testing correlates to a higher number of defects found in system level testing. In contrast, they refuted the hypothesis that a higher number of errors found in-house translates to a higher number of defects found by customers. An interesting statement by these researchers, is that it is difficult to state static software engineering laws, and that much of the results in these hypotheses and beliefs, appears to vary with the project. It is therefore interesting to continue studies such as this in our current case study, and observe whether particular patterns maintain constancy. This study is thought to be a valuable contribution to the empirical body of knowledge in software quality.

Other valuable work is found in the classic study by Adams [Ada84] which documented an extensive study on several IBM®products and found that the most pervasive failures in the field are those caused by defects found by customers just after the release of software products. These defects that cause repeated problems are relatively few in relation to the total number of defects that are eventually found. Adams was therefore able to motivate a process of preventive maintenance to preempt the redundant and therefore costly rediscoveries of these virulent problems by customers. In [POC93], the researchers conducted a measured study, this time to investigate the perceived usefulness of test coverage for assuring quality in large, realistic projects. The researchers were able to be more specific in their assertions and actually note the critical percentage level of coverage that was necessary for their project. Like [FO00], this relates to our work, as the extent to which testing is necessary is of importance to us.

Khoshgoftaar and colleagues [KAKG96] look at the issue of *fault-prone modules*. This is somewhat similar, though approached in a different manner, to [Ada84]. This study uses *discriminant analysis* [KAKG96] to define a *discriminant function* [KAKG96] that assigns modules to *fault prone* or *not fault prone* classes, and undertakes to prove the validity of these assignments. Their hypothesis is that being able to identify such a fault prone nature is enough to quantify the overall quality of the product, render-

ing fault prediction for the entire product unnecessary. Their most valuable finding was that this hypothesis, according to their study, is true. [SYTP85] provides very similar research and adds to our motivation to conduct additional studies into the relationship of in-house discovered defects and customer discovered defects. We will discuss much more of this related work as we investigate our own hypotheses and findings.

3 Terminology

This section briefly describes the terminology used in this paper, which includes definitions given in the glossary of the IEEE Standards Collection in Software Engineering [IEE91].

- Pre-release: Refers to any activity that occurs prior to the release of the software product.
- Post-release: Refers to any activity that occurs after the release of the software product.
- Testing: Refers to all testing studied in this research, such as regression and system testing.
- Defect: A variance from a desired product attribute. There are two types of defects: (1) defects from product specification occur when the product deviates from the specifications, and (2) defects from customer expectation occur when the product does not meet the customer's requirements, which failed to be expressed in the specifications.

  The development organization in this case considers a defect to be a defect from product specifications as the product is not custom-built and requirements are determined in-house.

  - The defect itself may mean any of the following: the specifications were implemented incorrectly, a specified requirement is missing, or there is an extra requirement that is included in the product that was not specified.
  - The testing team is able to determine a defect when a failure occurs and based on a specification document and product documentation. The customer is able to determine a defect when failure occurs and based on product documentation.
- Failure: Occurs when a defect causes an error in operation.
- Regression testing: Selective testing of a system component to verify that modifications have not caused unintended effects, and that the system or component still complies with its specified requirements. In the context of our case study, regression testing is composed of tests from the evolving functional testing repository. Functional testing is focused on the specified functional requirements and as such does not verify the interactions of system functions.
- System testing: Testing conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. This type of testing therefore evaluates the interactions of the functions in the environment specified in the requirements. In this case specifically, system testing is used to assess the overall quality of the code from the perspective of the customer. System testing is conducted in an environment that attempts to model as realistically as possible scenarios of customer usage. These scenarios are largely based on the quantitative knowledge of domain experts within the test team, as well as continuously gathered information from reported customer experiences with the product.
- In-house defects: Refers to all defects found by a test team for a product at the development organization's site prior to the release of that product.
- Field defects: Refers to all defects found by customers using the product in their organization post-release.
- Pearson correlation coefficient (PCC): The measure of the strength of the linear relationship between two random variables x and y. It is denoted by R. It ranges between 1 for a strong positive relationship, and -1 for a strong negative relationship [MI82]. Its square, R^2, indicates the strength of the fit of the data as it quantifies the sum of squares of deviation.

4 Our Hypotheses

The hypotheses we address can be categorized into two groups.

Group 1: Hypothesis for testing and its effect on pre-release defects

- The higher the percentage of the code exercised with all types of testing, and specifically regression and system testing, the more defects the in-house test team should find.

Group 2: Hypotheses for testing and its effect on post-release defects

- If more defects are found in a module by the in-house test team in pre-release testing, fewer defects will be found in that module by customers after the release of the software.

- If more coverage of a module is achieved through all types of testing, and specifically regression and system testing, fewer defects will be found by the customer.

5 The Data

5.1 The Product

The product used in this study is a large commercial product. We conducted the analysis in the context of system testing and regression testing. We were able to study 221 modules of the product, comprised of 201,629 lines of code (LOC). These 221 modules comprised two software components of the product. The code size is allocated to the modules as shown in Table 1. In these 221 modules the largest module was 7700 LOC, the smallest was 11 LOC, and the average size was 921 LOC. The product is highly complex and has been augmented through several versions over a period of many years. Our research examines the development period between two separately released *bundles* of software correction code, called *patches*. In addition to the corrective capability of these patches, they offer limited additional functionality. The defects to be corrected are those that have been reported by a few customers but not necessarily found by all customers. So the patches will preventively correct problems where they may not have been discovered yet.

5.2 The Metrics

In this development organization, when unspecified behaviour is discovered prior to release by the testing team, a report is generated for that defect. It is recorded in an official defect tracking system and reported to the development team for correction. A formal process is followed to handle such defect escalation through the development chain for correction, and subsequent rectification. This process may result in the generation of a corrective piece of code that is integrated into the system, at which time the defect is *closed*. A defect may also be identified as being *void* due to misunderstanding of the specifications for the product. Also when functionality included in the specification is noted to be missing from the product (prior to release) by the in-house test team, this omission may be *deferred* for some future release, and is counted as a pre-release in-house defect in the product. These are distinct from post-release field defects, found by the customer[1]. We only consider those defects that are closed or deferred. These are *valid* defects. When identical defective behaviour is identified on more than one occasion, the defect is considered to be a *duplicate* and is not counted.

Let us call the previous version of the product X, and the version by which we contextualize pre- and post-release Y. Product Y was the most recent release at the time of the study. The parameters that define this study for product Y are the following: the code coverage achieved by system testing, and the code coverage achieved through regression testing. These are the independent variables [Per00]. The dependent variable in this case is the defect. This includes both in-house pre-release defects for Y and post-release field defects. These defects are the perceived end product of the overall testing process.

Code coverage was measured using the software testing and analysis tool – ATAC (Automatic Test Analysis for C or C++) [ML94, Tel98]. (The code for the commercial product is C code based.) Each of the 221 modules has corresponding levels of coverage that were achieved by running regression and system level testing on them. They each also have a corresponding number of field and in-house defects associated with them. This facilitates analysis by allowing us to plot graphs of coverage percentage values against defects for all modules. In section six we conduct such analysis by plotting coverage percentage values on the x-axis (in both regression and system testing cases), and field defects and in-house de-

[1] It is important to note that because those items that fall into the deferred category are functionality based, when deferral decisions are made, these items are not included in the instructional documentation for the customer. As a result, when the product is released, customers are unaware of deferred functions, thus they cannot consider missing functionality that is not documented to be a valid field defect.

Size (Lines of Code)	No. of Modules
up to 500	107
501 – 1000	56
1001 – 2000	34
2001 – 3000	14
3001 – 4000	4
4001 – 5000	2
5001 – 6000	1
6001 – 7000	1
7001 – 8000	2
Total	221

Table 1: Modules and Lines of Code

fects on the y-axis for all files. In the histogram however, instead of creating one point for each coverage value to defect relationship, we total all defects for all files with coverage recorded within categories of ten percentage values and plot them according to this grouping. This displays the trends of the defects according to the code coverage categories. This is explained further in section six.

The coverage analysis is based on data flow through the program and we were able to collect a coverage value for each module, for both types of testing, using the following procedure.

1. The program was prepared for testing with a preprocess-compile-link phase. In this step, running the ATAC tool, instrumented the source code, and tables outlining data flow were generated. This instrumentation and the tables are used at product code run-time by ATAC.

2. The system and regression test scripts were run with the product code. The ATAC run-time program collected trace and coverage information.

3. The coverage and trace information provided coverage values for the tests conducted. It also offered information on the source code constructs that were not covered by the tests.

ATAC offers eight types of coverage [Tel98]. The simplest is *function entry coverage*, which is an analysis of coverage that ensures that all functions within a program are called at least once. A more powerful level of coverage is *branch-free* or *block* coverage, which offers a slightly stronger version of coverage by analyzing the coverage of basic blocks[2]. The most involved type is *all-uses* coverage, which ensures that when a variable can be assigned in several ways, each assignment is tested. We used block coverage for our study as it is a common coverage measure for industrial analysis [GWTH98]. We will consider other types of coverage in future work.

The achievement of 100% block coverage through testing, verifies that all *basic blocks* have been executed at least once [Tel98]. So, the instructions in any basic block are either executed all together, or not at all. In C and C++, a block may contain more than one statement if no branching occurs between statements. A statement may contain multiple blocks if branching occurs inside the statement. An expression may contain multiple blocks if branching is implied within the expression (for example, conditional, logical-and, and logical-or expressions). ATAC begins a new basic block after a function call because it is possible that the function call will not return (for example if exit is called within the function)[Tel98].

The in-house defects and field defects associated with each module were collected from the defect tracking system by carrying out a query for:

1. Valid defects reported in-house for the product between the release dates of product X and product Y.

2. Valid defects reported by customers for field defects for the product after the release date for product Y.

We are aware that some of the recently reported in-house and field defects may have flowed from defects in previous versions of the product. These defects may only be discovered now as part of the product labelled Y. However, as these defects were not previously revealed in any earlier use, and we do not consider duplicate defects to be valid, it is satisfactory to

[2] A basic block, or simply, a block, is a code sequence that is always executed sequentially, meaning, it has no internal branching constructs. It is also described as any "single-entry-single-exit" region of code [Tel98]

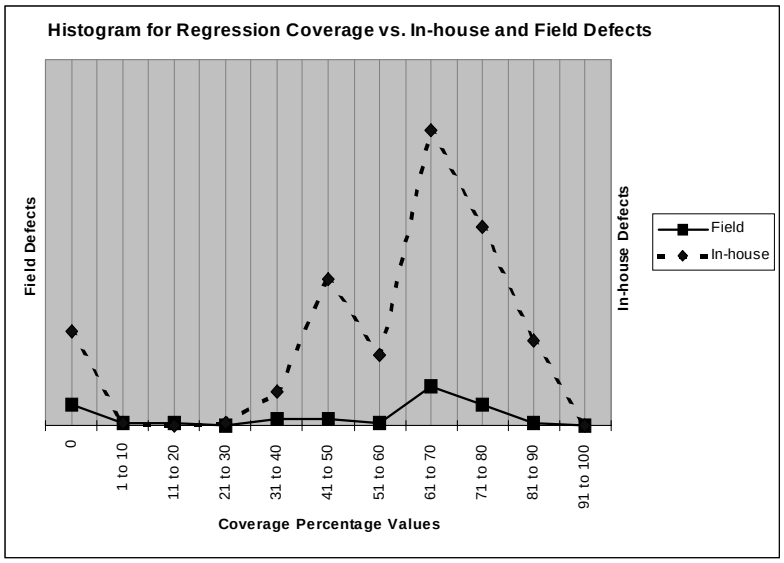

Figure 1: Analysis of Regression Coverage and Defects

group these with all other defects for Y. This is also a reasonable approach because in a highly quality controlled environment, such as the release environment for the product under study, no known defects are released to the customer, so once again any field defects would not have been previously revealed by the in-house team.

6 The Hypotheses and Associated Results

This is a comparative discussion, this means that we are comparing belief with fact. The belief that we are challenging is both our own and the belief of others. Consequently, it is important to furnish the reader with some background on how the hypotheses that we discuss in this paper are presented by other researchers. We will briefly present that information here as part of the exploration of our analysis and findings.

6.1 Hypothesis 1

The perception by those in software quality research and practice is that when more code is exercised through testing, more defects should be found by the in-house test. Kit [Kit95] notes an example of this. He states the following for statement level coverage which is equivalent to block coverage:

> *The probability of finding new defects is inversely proportional to the amount of code not yet covered. The closer we are to achieving 100% coverage, the more likely it becomes that we are searching in previously unnavigated territory and the more likely it is that we will find more defects per line of code.*

We therefore present such a belief in Hypothesis 1a .

Hypothesis 1a: The higher the percentage of the code exercised with all types of testing, and specifically regression and system testing, the more defects the in-house test team should find.

In Figures 1 and 2 we present histograms of the behaviour of defects[3] found by the in-house testing team for the 221 modules when mapped against the level of

[3]We must note for the purpose of clarity, that these defects are **all** of those extracted for the system between the release of the two bundles. We did not distinguish between defects revealed during regression testing or system level testing. After this defect extrac-

coverage achieved in groups of 10 percentage values for those modules. For example, there are three modules for which coverage was achieved at levels varying from 11% to 20%. We totaled the defects that were found in these three modules by the in-house testing team, running regression tests prior to the release of Y. This total is presented as a single point on the in-house defect line in Figure 1 for 11-20% regression coverage.

It is very important for us to point out at this time that our histograms are plotted with the same scale for field defects and in-house defects. Meaning, that if we plot a point (a, b) for field defects vs. coverage percentage values, it will fall on the same horizontal plane of a point (c, b) for in-house defects. We have removed the actual values on the vertical axes of the graphs due to the the classified nature of the raw defect data. This must be emphasized as we placed field defects and in-house defects on separate axes merely for visual clarity and not because of differing scales.

We have also chosen to include the post-release field defects on the graphs of in-house pre-release defects. This allows the relationship between the two graphs to reveal itself immediately. We will reserve the discussion of the post-release observations for the next section. For further comparison we show, the two graphs, Figures 1 and 2 merged into a single graph in Figure 3. This allows us to observe the similarity between the two graphs. We see that both pre-release in-house defects and post-release field defects (as correlated to system and regression coverage) show similar trends.

It is important to note for Figure 3, that because this is generated for the sake of comparison, the defects referred to as regression field defects and system field defects are distinguished between the types of coverage even though they are the same defects. This is because they are correlated to regression test coverage and system test coverage respectively. This therefore means that the defects are sorted according to the type of coverage that they are correlated to, and so maintain their graphical shapes from Figures 1 and 2. So the x-axis shows a label of *Coverage Percentage Values*, which in this case includes regression and system test coverage.

Examining these graphs we notice that the perception of more testing resulting in the discovery of more defects up to the maximum of 100% is not substantiated in this case. The unexpected relationship shows itself for the regression test cases and for the system test scenarios. The observation here is one of effectiveness of testing up to a point at which the productivity of testing efforts appears to decline. This peak value for coverage achievement occurs for these modules in the 61% to 70% range when correlated to regression test coverage (Figure 1), and the 51% to 60% range when correlated to system test coverage (Figure 2). Piwowarski, Ohba and Caruso [POC93] also observed a similar trend of a decline after some peak value for in-house defects for their project, and stated that in their case " *70% statement coverage is the critical point for their function test to ensure that the test cases sufficiently exercise and cover all output patterns*". We must also emphasize the relative flatness of the field defect graphs with respect to the height of the graphs for the defects found by the in-house test team. This is definitely a sign of strong productivity on the part of the in-house test team.

6.2 Hypothesis 2

Our second set of hypotheses relates to breadth of testing and its effects on post-release defects. The widely held belief is that the more testing an organization does, the better the product the customers receive will be with respect to fewer defects. This larger view is broken up into two specific hypotheses.

Hypothesis 2a: If more defects are found in a module by the in-house test team in pre-release testing, fewer defects will be found in that module by customers after the release of the software.

This is an interesting hypothesis and is the foundation for the primary objective of most testers. The statements of this hypothesis are largely anecdotal. Pressman [Pre97] states that if testing is conducted successfully, with the objective of offering a high probability of finding undiscovered errors, there will be a *good indication* that functions are working according to specification, that is, without defect. This is qualified by Pressman by including the fact that, to achieve such a *good indication*, there must be continuation in the testing process to actually reveal such defects. He also premises his statements by adding that it is impossible to show that there are no defects remaining in a product, but only to show that some defects are present.

tion, we then correlated the numbers of all defects with the measured coverage values in order to see what impact the levels of testing had on the overall quality of the product. We show these defects in a histogram format as explained in section 6.1 .

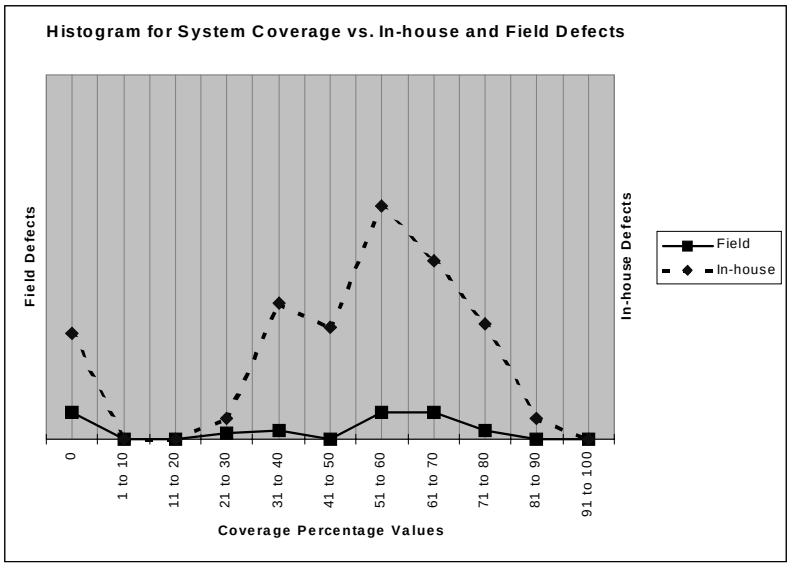

Figure 2: Analysis of System Coverage and Defects

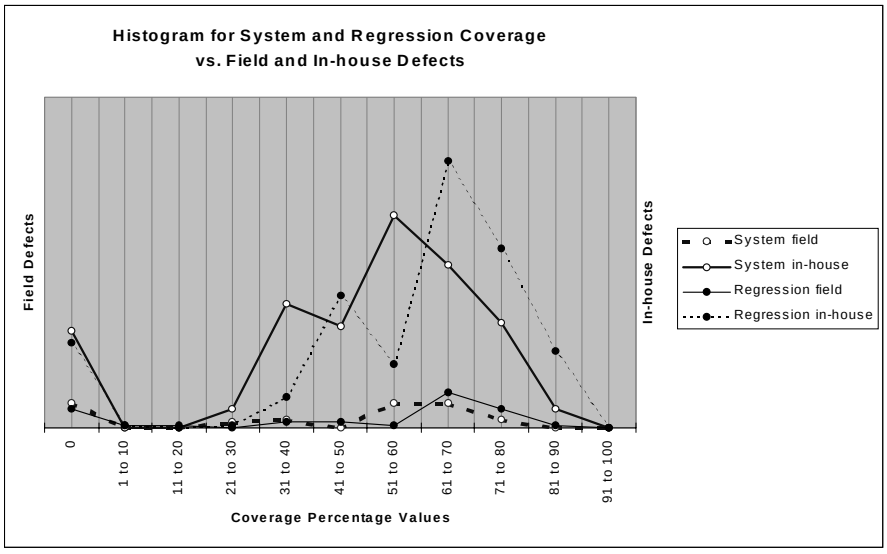

Figure 3: Analysis of Regression and System Coverage and Defects

He observes that, "*…as test results are gathered and evaluated, a qualitative indication of software quality and reliability begins to surface*." This means that, the qualitative belief of test organizations that the discovery of more defects by their efforts prior to release, indicates that the customer will be faced with fewer defects while the product is in the field.

Figure 4 is a column and line graph of the pre-release (in-house) defects found in the modules, and the post-release (field) defects found in the same modules, with in-house defects represented by the grey area on the right side of the graph, and field defects represented by the narrow, dark columns. These are plotted against a listing of the modules ordered from left to right for those modules with the least amount of pre-release defects to those with the most pre-release defects. This graph shows the in-house and field defects for each file as fractions of the overall total number of in-house and field defects. For example, if two pre-release defects in module c_1 of component C were found, and the overall total of all defects in all modules $\{c_1, ..., c_n\}$ in component C was 10, then the value for c_1 that would show on the graph is 0.2. This allows us to present the scale of the trends of the defect values relative to each other and maintain the classified nature of the raw data.

The Pearson correlation coefficient (PCC), of the strength of the linear relationship between field and in-house defect types, was calculated. The relationship was a weak positive relationship with correlation coefficient value 0.3837. The hypothesis is therefore not supported. We must also offer the reader the R^2 value, as this is a necessary measure of how well the linear model fit the data. An R^2 value of zero implies a complete lack of fit of the linear model to the data, and a value of one implies a strong fit. In this case the R^2 value is 0.1472. So we are seeing a very weak relationship where the modules in which more errors tend to be found by the in-house test team, have a tendency to be the same modules where proportionately more errors are found by the customers.

In the empirical study done by Fenton and Ohlsson [FO00], they hypothesized that a higher incidence of faults in all pre-release testing (which in their study consists of system testing and functional testing), implies higher incidence of faults in operation. Their belief was founded in a perception of the existence of *rogue modules*. These are the perceived small number of modules that account for most of the defects occurring in code, thus presenting themselves as defect infested both before and after release. They found no evidence in their data to support this belief. We have, on the contrary, observed a contradiction of our own hypothesis on this topic, and some implication of the hypothesis observed by Fenton and Ohlsson [FO00] for our product. This will have to be studied further to observe more definitive evidence.

Hypothesis 2b: If more coverage of a module is achieved through all types of testing, and specifically regression and system testing, fewer defects will be found by the customer.

This hypothesis is strongly related to the previous one. This implies that the more rigorous and complete one's efforts are in testing, the higher quality the final product will be. Lyu, Horgan, and London [ML94] hypothesize that good data flow testing produces a good coverage score, and good data flow testing implies good software. They conducted investigations of the usefulness of the ATAC tool to test flight control software, with this hypothesis as one of its gauges. Their experience was that high coverage translated to lower defects in the field. In fact, the researchers only encountered one defect in the field.

In our analysis there appears to be a non-linear relationship between post-release defects and module coverage. This can be seen in Figures 1-3. The relationship is very similar to the relationship for pre-release defects and module coverage. However, the counterintuitive discovery here is that, as the amount of pre-release testing increases in a module, so do the number of defects found by the customer up to approximately 61-70% coverage. These field defects decline after this point and then we see the expected field defect behaviour. This expected behaviour is a decline in the number of field defects found by customers in modules, as in-house test coverage for those modules increases in the ranges from 71% to 100%.

We do not claim at this point, that this is a general trend to be expected in the testing of every software product. We do however see a logical explanation for this finding in our belief that there is a level of testing for which we must strive, and only after we have achieved this level of testing can we be confident about the expectation of our hypothesis for increased quality. We also find the similarity of the curves for pre-release and post-release defects in relation to testing coverage quite interesting, and note that both of these curves seem to indicate that level of testing that is optimal as in [POC93]. We will continue to investigate this relationship in further work.

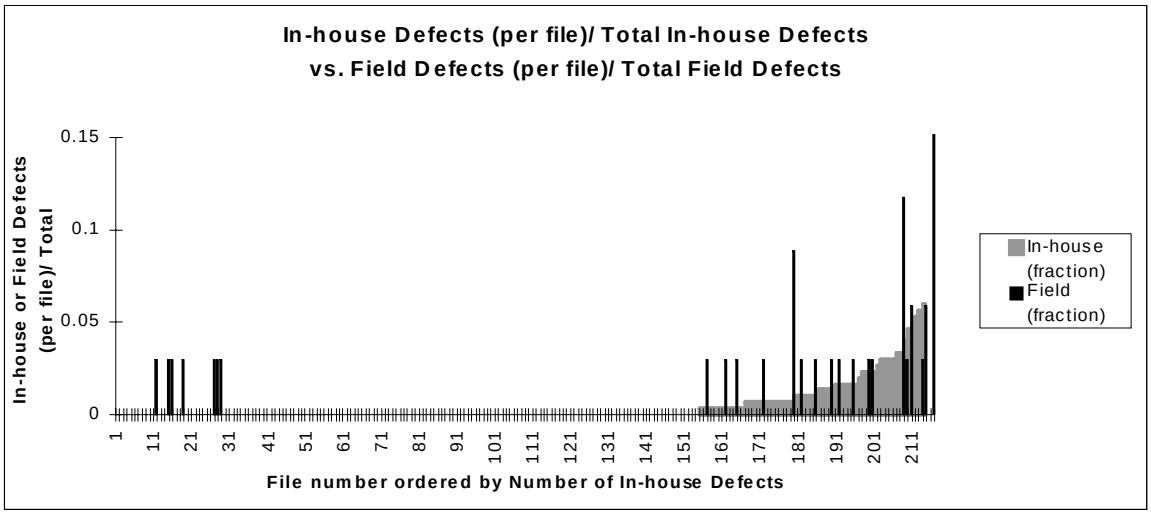

Figure 4: Analysis of In-house Defects and Field Defects

7 Conclusions and Future Work

In this study of the relationships between defects and testing, we have made some interesting observations. We have seen that in the effort to find as many of the defects hidden in a product as possible, the testing method that exercises 100% of the product may be spending too many resources on the testing effort. It is actually the case that testing is optimized to a point for block coverage, and any other efforts spent on in-house testing will be less productive in their defect yield prior to product release. This confirmed and added to work in [POC93]. We also saw that with increased coverage for modules, the defects that customers see in the product also increases to a point, and then declines. This implies that there is a level of coverage that the testing organization must insist on achieving before a stop-test decision can be considered. In our case that critical coverage range that must be achieved is 71-80%. It also implies that the code that is being covered is not always the code where the customers are finding the defects, so the code is not always being exercised in-house in a manner that is similar to field use.

Furthermore, if the two results are considered in conjunction, we note that apparent productivity in testing in-house does not mean that there will be more satisfaction in the field. Rather, the testing must be effectively targeted. This observation is actually borne out in our investigation of the correlation between the number of defects found in-house pre-release, to the number of defects found by the customer after release. This relationship did not have the strength that we originally believed that it would. It was weakly positive, that is, there was some tendency for the modules where more defects are found by the pre-release test team to coincide with the modules where the customer finds more defects. This therefore complements the findings of [KAKG96], where the researchers state that the most fault-prone modules indicate the field quality of the software. We must state that our weakly positive relationship was not a strong linear fit, but we can state that our prior belief that in-house efforts would result in strong field quality was not as obvious as first believed. Once again, we see that more effort, and even more in-house yield, does not necessarily correlate with more quality.

We should also note that in this case there were a few files in which there were field discovered defects but no in-house discovered defects. If we look closely at Figure 4, this must be taken into consideration, as it may have implications for the data of Figures 1 to 3. They show that even with increased coverage both in regression and system testing, customers are still finding an increasing number of defects to a critical point. Thus, the in-house testing, though highly pro-

ductive for most files, is overlooking a few defects in others, as seen in the field activity in Figure 4. We have noted previously that there is quite a large separation between the pre-release and post-release defects in Figures 1 to 3, which indicates high effectiveness by the test organization. We see this as an opportunity for enhanced efforts in ensuring that potential customer use of the system is taken into consideration and such field defect activity where there is no in-house defect activity is eliminated. This adds to the previous findings of [Ada84] as he has found that customer behaviour tends to reveal overlooked latent defects. This also encourages us to explore the relationship between in-house system testing that is focussed on customer usage scenarios, and observe the resulting relationship of in-house defects to field defects, as well as the relationships revealed between coverage and this kind of testing. This will be future work.

We do not claim that these observations can be generalized for all projects, as they were made for two bundles of a product which is much larger in comparison, and consists of much more extensive functionality. We do believe that the observations are valuable as an addition to the existing documented knowledge. In this study we chose to do pairwise comparisons between variables. In addition to the usage-based testing analysis mentioned, we intend to conduct additional studies in this analysis to investigate multivariate relationships. An example of the usefulness of this extended exploration would be correlations of the relationships between code coverage, in-house defects, and field defects. In such a study, we would explore the conjunction that was included in our discussion above. This would allow us to further quantify how testing and in-house defect revelation impacts on product quality. The ultimate result might be a regression model that presents a function that allows us to see what the optimal amount of testing and coverage would be to achieve an acceptable level of post-release defects. This would be a risk analysis model. Such potential models offer us an indication of how powerful empirical research such as this can be in possible statistical process control. As a result, we see the possibilities for a more precise practice of software quality, and on a more general level, we see possibilities for the growth of software engineering.

Appendix

At the time of the work conducted in this paper, ATAC was the licensed version of the product utilized for coverage measurement. For the purpose of reference, and to ensure that all information is current, it should be noted that the product ATAC has since been repackaged and is now marketed by the company Cleanscape under arrangement with Telcordia Applied Research. ATAC has been renamed TestWise.

Acknowledgements

The authors would like to offer our appreciation to the anonymous referees whose recommendations have assisted with the final presentation of this work. We would also like to acknowledge the support from the Centre for Advanced Studies (CAS) for this collaboration between IBM®and the University of Western Ontario.

About the Authors

Mechelle Gittens has worked and carried out research in the area of Computer Science since 1995. For six years within this time she has undertaken full-time graduate studies and research at the University of Western Ontario (UWO), first in the Masters of Science in Computer Science, and subsequently in the Doctoral program in Computer Science. She is published in the area of Software Quality, and has particular interests in Software Reliability, Software Project Management, Software Testing, and Empirical Software Engineering. She has also taught and consulted for the Institute for Government Information Professionals in Ottawa.

Her research has led to work with the IBM Centre for Advanced Studies for research into Customer Based Testing. She has also conducted research with Northern Telecom for research into Defect Prediction and Software Estimation. She has acted in the capacity of President of the Kitchener Waterloo Software Quality Association (KWSQA), and is currently the Education Officer for this group of software quality professionals. She holds Bachelor and Masters of Science Degrees in Computer Science, and is expected to complete her Ph.D. in Fall 2002. She is also a member of the Association for Computing Machinery (ACM),

as well as the Southwestern Ontario Software Quality Group (SWO SQG).

Hanan Lutfiyya is an associate professor of Computer Science at the University of Western Ontario. She received her Ph.D. from the University of Missouri at Rolla in 1992. Her research interests are distributed systems and software engineering with a specific focus on application reliability.

Michael Bauer is a Professor of and Chair of the Department of Computer Science at the University of Western Ontario. He was also Chair of the Department from 1991-1996. From 1996-2001 he was the Associate Vice-President Information Technology at the University of Western Ontario. He was the Executive Director of LARG*net, a not for profit company operating and developing a metropolitan/regional network in and around London, Ontario.

He is a member of the IEEE and the Association for Computing Machinery (ACM) and has served on various committees of both organizations. He has also served on the organizing and program committee of numerous conferences including, the International IEEE Workshop on Systems Management, the International Conference on Open Distributed Processing, IEEE Computers, Systems and Applications Conference and has refereed for a variety of international journals, including IEEE Computer and the Communications of the ACM.

His Ph.D. is in Computer Science is from the University of Toronto; he also holds B.Sc. and M.Sc. degrees in Computer Science. He has published over 100 papers on various topics in the computing field and has collaborated with a number of companies on joint research projects. His research interests include distributed computing, particularly the management of distributed applications and systems, network management, software engineering, and high performance computer networks. He has also been the recipient of The University of Western Ontario's Excellence in Teaching Award.

David Godwin graduated from York University in 1985 with his Bachelor of Science. Since joining IBM in 1989, he has held positions in System Verification Test (SVT), and in product service. He started with IBM as a testing contractor and subsequently accepted a full-time position with IBM in 1989. He held the position of Team Lead for system testing until 1996, and continued his career with IBM in the roles of Service Analyst and Team Lead from 1996 to 2000. In 2000, he returned to System Verification Testing and is presently a Quality Assurance (QA) Manager responsible for system testing for three major departments within IBM.

Yong Woo Kim holds a Master's degree in Computer Science from Florida Institute of Technology, USA. He has been involved in studying efficient use of code coverage in large size software development. His research interests include: Software Cost Modelling, Software Testing, and Large Size Software Development. Currently, Yong works in System Verification Testing at IBM.

Pramod Gupta obtained his B.Sc. (Physics) degree in 1991 (Gujarat University, India), M.A. (Physics) degree in 1995 (University of Rochester, New York), Ph.D. (Physics) degree in 1999 (University of Rochester, New York). From late 1998 to early 2000, he was a research associate in the Department of Physics at McMaster University, Ontario. In early 2000, he joined IBM where he has worked on software testing tools, automation and code coverage.

References

[Ada84] Edward N Adams. Optimizing preventive service of software products. *IBM Journal of Research*, 28(1), January 1984.

[Das92] M.K. Daskalantonakis. A practical view of software measurement and implementation experiences within motorola. *IEEE Transactions on Software Engineering*, 18(11), November 1992.

[FO00] Norman E. Fenton and Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8), August 2000.

[GWTH98] Swapna S. Gokhale, W. Eric Wong, Kishor S. Trivedi, and J. R. Horgan. An analytical approach to architecture-based software reliability prediction. In *Proceedings of the 3rd International Performance and Dependability Symposium*, URL: citeseer.nj.nec.com/article/gokhale98analytical.html, 1998.

[IEE91] IEEE. *IEEE Standard 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology*, February 1991.

[KAKG96] T. M. Khoshgoftaar, E.B. Allen, K.S. Kalaichelvan, and N. Goel. Early quality

prediction: A case study in telecommunications. *IEEE Software*, 26(8), January 1996.

[Kit95] Edward Kit. *Software Testing In the Real World: Improving the Process*. ACM Press., 1995.

[MI82] James McClave and Frank H. Dietrich II. *Statistics - Second Edition*. Dellen Publishing, 1982.

[ML94] Saul London Michael Lyu, J.R. Horgan. A coverage analysis tool for the effectiveness of software testing. *IEEE Transactions on Reliability*, 43(4), December 1994.

[NF94] Robert L. Glass Norman Fenton, Shari Lawrence Pfleeger. Science and substance: A challenge to software engineers. *IEEE Software*, 11(4), July/August 1994.

[Per00] William E. Perry. *Effective Methods for Software Testing*. John Wiley and Sons, Inc., 2nd edition, 2000.

[POC93] P. Piwowarski, M. Ohba, and J. Caruso. Coverage measurement experience during function test. In *Proceedings of the 15th International Conference on Software Engineering*. IEEE CS, 1993.

[PPV00] D. Perry, A. Porter, and L. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the 22nd Conference on Software Engineering*, Limerick, Ireland, June 2000.

[Pre97] Roger Pressman. *Software Engineering: A Practitioners Approach*. McGraw-Hill, 4th edition, 1997.

[SYTP85] V. Shen, T-J Yu, S. Thebaut, and L. Paulsen. Identifying error-prone software – an empirical study. *IEEE Transactions on Software Engineering*, 11(4), 1985.

[Tel98] Telcordia. Telcordia software visualization and analysis toolsuite (xsuds). http://xsuds.argreenhouse.com/html-man/, July 1998.

[TG96] A.A. Takang and P.A. Grubb. *Software Maintenance: Concepts and Practice*. Thomson International Computer Press, 1996.