High-Impact Defects: A Study of Breakage and Surprise Defects

Emad Shihab
Software Analysis and Intelligence Lab (SAIL)
Queen's University, Canada
emads@cs.queensu.ca

Audris Mockus
Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

Yasutaka Kamei, Bram Adams and Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University, Canada
{kamei, bram, ahmed}@cs.queensu.ca

ABSTRACT

The relationship between various software-related phenomena (e.g., code complexity) and post-release software defects has been thoroughly examined. However, to date these predictions have a limited adoption in practice. The most commonly cited reason is that the prediction identifies too much code to review without distinguishing the impact of these defects. Our aim is to address this drawback by focusing on *high-impact defects* for customers and practitioners. *Customers* are highly impacted by defects that break pre-existing functionality (breakage defects), whereas *practitioners* are caught off-guard by defects in files that had relatively few pre-release changes (surprise defects). The large commercial software system that we study already had an established concept of breakages as the highest-impact defects, however, the concept of surprises is novel and not as well established. We find that surprise defects are related to incomplete requirements and that the common assumption that a fix is caused by a previous change does not hold in this project. We then fit prediction models that are effective at identifying files containing breakages and surprises. The number of pre-release defects and file size are good indicators of breakages, whereas the number of co-changed files and the amount of time between the latest pre-release change and the release date are good indicators of surprises. Although our prediction models are effective at identifying files that have breakages and surprises, we learn that the prediction should also identify the nature or type of defects, with each type being specific enough to be easily identified and repaired.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Complexity measures, Performance measures*

General Terms

Software Quality Assurance

Keywords

High-Impact, Process Metrics, Defect Prediction

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

1. INTRODUCTION

Work on defect prediction aims to assist practitioners in prioritizing software quality assurance efforts [6]. Most of this work uses code measures (e.g., [25, 35]), process measures (e.g., [11]), and social structure measures (e.g., [4]) to predict source code areas (i.e., files) where the post-release defects are most likely to be found. The number of pre-release changes or defects and the size of an artifact are commonly found to be the best indicators of a file's post-release defect potential.

Even though some studies showed promising performance results in terms of predictive power, the adoption of defect prediction models in practice remains low [10, 13, 29]. One of the main reasons is that the amount of code predicted to be defect-prone far exceeds the resources of the development teams considering inspection of that code. For example, Ostrand *et al.* [27] found that 80% of the defects were in 20% of the files. However, these 20% of the files accounted for 50% of the source code lines. At the same time, all defects are considered to have the same negative impact, which is not realistic, because, for example, documentation defects tend to be far less impacting than security defects. At the same time, model-based prediction tends to indicate the largest and the most changed files as defect-prone: areas already known to practitioners to be the most problematic.

In this work, we address the problem of predicting too many defects by focusing the prediction on the limited subset of *high-impact defects*. Since impact has a different meaning for different stakeholders, we consider two of the possible definitions in this paper: *breakages* and *surprises*. Breakages are a commonly accepted concept in industry that refer to defects that break functionality delivered in earlier releases of the product on which customers heavily rely in their daily operations. Such defects are more disruptive because 1) customers are more sensitive to defects that occur in functionality that they are used to than to defects in a new feature and 2) breakages are more likely to hurt the quality image of a producer, thus directly affecting its business. To ensure that we focus only on the highest impact defects, we only consider breakages that have severities "critical" (the product is rendered non-operational for a period of time) and "high" (the product operation has significant limitations negatively impacting the customer's business).

Whereas breakages have a high impact on customers, surprise defects are a novel concept representing a kind of defects that highly impact practitioners. Surprises are defects that appear in unexpected locations or locations that have a high ratio of post-to-pre release defects, catching practitioners off-guard, disrupting their already-tight quality assurance schedules. For example, post-release

defects in files that are heavily changed or have many defects prior to the release are expected and scheduled for. However, when a defect appears in an unexpected file, the workflow of developers is disrupted, causing them to vacate their current work and shift focus to addressing these surprises.

The investigated project has used the concept of breakages for several decades but surprise defects are a new concept. Hence, we start by comparing the properties of breakage and surprise defects and qualifying the impact of surprises. We then build prediction models for breakages and surprise defects (RQ1), identify which factors are the best predictors for each type of defect (RQ2) and quantify the relative effect of each factor on the breakage and surprise-proneness (RQ3). Finally, we calculate the effort savings of using specialized defect prediction models and perform a qualitative evaluation of the usability of the prediction models in practice and propose ways to make such prediction more relevant.

We make the following contributions:

- **Identify and explore breakage and surprise defects.** We find that breakage and surprise defects are approximately one-fifth of the post-release defects. Only 6% of the files have both types of defects. For example, breakages tend to occur in locations that have experienced more defect fixing changes in the past and contain functionality that was implemented less recently than the functionality in locations with surprise defects.

- **Develop effective prediction models for breakage and surprise defects.** Our models can identify future breakage and surprise files with more than 69% recall and a two to three fold increase of precision over random prediction.

- **Identify and quantify the major factors predicting breakage and surprise defects.** Traditional defect prediction factors (i.e., pre-release defects and size) have a strong positive effect on the likelihood of a file containing a breakage, whereas the co-changed files and time-related factors have a negative effect on the likelihood of a file containing a surprise defect.

- **Measure the effort savings of specialized prediction models.** Our custom models reduce the amount of inspected files by 3-30%, which represents a 21-24% reduction in the number of inspected lines of code.

- **Propose areas and methods to make defect prediction more practical.** A qualitative study suggests that an important barrier to the use of prediction in practice is lack of indications about the nature of the problem or the ways to solve it. The method to detect surprise defects may be able to highlight areas of the code that have incorrect requirements. We propose that an essential part of defect prediction should include prediction of the nature of the defect or ways to fix it.

The rest of the paper is organized as follows. Section 2 highlights the related work. Section 3 compares the properties of breakages and surprise defects. Section 4 outlines the case study setup and Section 5 presents our case study results. Section 6 discusses the effort savings provided by the specialized models built in our study. Section 7 highlights the limitations of the study. Section 8 reflects on the lessons learned about the practicality of defect prediction and details our future work. We conclude the paper in Section 9.

2. RELATED WORK

The majority of the related work comes from the area of defect prediction. Previous work typically builds multivariate logistic regression models to predict defect-prone locations (e.g., files or directories). A large number of previous studies use complexity metrics (e.g., McCabe's cyclomatic complexity metric [19] and Chidamber and Kemerer (CK) metrics suite [5]) to predict defect-prone locations [3, 12, 24, 26, 32, 35]. However, Graves *et al.* [11], Leszak *et al.* [18] and Herraiz *et al.* [15] showed that complexity metrics highly correlate with the much simpler lines of code (LOC) measure. Graves *et al.* [11] argued that change data is a better predictor of defects than code metrics in general and showed that the number of prior changes to a file is a good predictor of defects. A number of other studies supported the finding that prior changes are a good defect predictor and additionally showed that prior defect is also a good predictor of post release defects [1, 14, 17, 18, 23, 33]. To sum up, this previous work showed that complexity metrics and hence size measured in LOC, prior change and prior defects are a good predictor of defect-proneness. The focus of the aforementioned work was to improve the prediction performance by enriching the set of metrics used in the prediction model. In this work, we incorporate the findings of previous work by using traditional defect prediction metrics/factors, in addition to other more specialized factors related to co-change and time properties to predict the location of highly impacting defects. However, we would like to note that our focus here is capturing high-impact defects rather than adding metrics to improve the performance of defect prediction models in general.

Although it has been shown that defect prediction can yield benefit in practice, its adoption remains low [10, 13]. As Ostrand *et al.* [27, 28] showed, 20% of the files with the highest number of predicted defects contain between 71-92% of the defects, however these 20% of the files make up 50% of the code. To make defect prediction more appealing to practice, recent work examined the performance of software prediction models when the effort required to address the identified defect is considered (i.e., effort-aware defect prediction) [2, 16, 20]. Although effort is taken into account, these models still predict the entire set of post-release defects, giving each defect equal impact potential.

Other work focused on reducing the set of predicted defects based on the semantics of the defect. For example, Shin *et al.* [31] and Zimmermann *et al.* [34] focused on predicting software vulnerabilities since they have high priority. Instead of narrowing down the set of defects vertically based on the semantics of the defects, we narrow down the defects horizontally across domains. For example, our models can predict high impact defects across many domains, whereas a model focused on vulnerability defects is only useful for one domain.

3. BREAKAGE AND SURPRISE DEFECTS

In this section, we provide background of the software project under study, and define breakages and surprise defects. We then characterize and compare breakage and surprise defects.

3.1 Background

Software Project: The data used in our study comes from a well-established telephony system with many tens of thousands of customers that was in active development for almost 30 years and, thus, has highly mature development and quality assurance procedures. The present size of the system is approximately seven million non-comment LOC, primarily in C and C++. The data used in our study covers five different releases of the software system.

Change Data: There are two primary sources of data used. Sablime, a configuration management system, is used to track Modification Requests (MR), which we use to identify and measure the size of a software release and the number of defects (pre-release and post-release). When a change to the software is needed, a work item (MR) is created. MRs are created for any modification to the code: new features, enhancements, and fixes. The project uses internally developed tools on top of the Source Code Control System (SCCS) to keep track of changes to the code. Every change to the code has to have an associated MR and a separate MR is created for different tasks. We call an individual modification to a single file a delta. Each MR may have zero or more deltas associated with it. Since the development culture is very mature, these norms are strictly enforced by peers.

For each MR, we extracted a number of attributes from Sablime and the SCCS: the files the MR touches, the release in which the MR was discovered, the date the MR was reported, the software build where the code was submitted, the resolution date (i.e., when the MR was fixed/implemented), resolution status for each release the MR was submitted to, the severity and priority of the MR, the MR type (e.g., enhancement or problem) and the general availability date of the release that includes the MR.

For each release, we classify all MRs into two types: pre-release changes (or defects if they are type problem), and post-release defects. MRs that are submitted to a release before the General Availability date (we refer to it as GA), are considered to be pre-release defects. Fixes reported for a particular release after the GA date, are considered to be post-release defects.

3.2 Defining Breakages and Surprise Defects

Breakage Defects: Defects are introduced into the product because the source code is modified to add new features or to fix existing defects. When such defects break, i.e., cause a fault or change existing functionality that has been introduced in prior releases, we call these defects breakages. The concept of breakages typically is familiar in most companies. For example, in the project studied in this paper, all severity one and two defects of established functionality are carefully investigated by a small team of experts. In addition to a root-cause analysis and suggestions to improve quality assurance efforts, the team also determines the originating MR that introduced the breakage. While other companies may use a different terminology than "breakages", most investigate such high-impact problems just as carefully, therefore similar data is likely to be available in other projects with mature quality practices.

Surprise Defects: Previous research has shown that the number of pre-release defects is a good predictor of post-release defects (e.g., [23, 35]). Therefore, it is a common practice for software practitioners to thoroughly test files with a large number of pre-release defects. However, in some cases, files that rarely change also have post-release defects. Such defects catch the software practitioners off-guard, disrupting their already-tight schedules. To the best of our knowledge, surprise defects have not been studied yet, and therefore are not recorded in issue tracking systems. Hence, for the purpose of our study, we use one possible definition of surprise. We define the degree to which a file contains surprise defects as:

$$Surprise(file) = \frac{No.\ of\ post\ release\ defects(file)}{No.\ of\ pre\ release\ defects(file)},$$

whereas in files without pre-release defects we define it as:
$Surprise(file) = No.\ of\ post\ release\ defects(file) * 2.$

Because our definition of surprise is given in terms of the ratio of post-to-pre release defects, we need to determine from what threshold the ratio should be considered significant. For example, if a file has one post-release defect and 10 pre-release defects, i.e., the defined surprise value is $\frac{1}{10} = 0.1$, should this file be flagged as being a surprise?

To calculate the surprise threshold, we examine all the files that had a defect reported within one month of the GA date of the previous release. The intuition behind this rule is that high impact defects are likely to be reported immediately after the software is released. Hence, we calculate the median of the ratio of all post-release defects and all pre-release defects for all files changed within one month of the previous GA date, then use this value as the surprise threshold for the next release. For example, the surprise threshold for release 2.1 is determined by the median ratio of post-to-pre-release defects of all files that had a post-release defect reported against them within a month after release 1.1.

3.3 Occurrence of Breakage and Surprise Defects

Before analyzing the characteristics of breakage and surprise defects, we examine the percentage of files that have breakages and surprise defects. To put things in context, we also show the percentage of files with one or more post-release defects. Table 1 shows that on average, only 2% of the files have breakages or surprise defects. That is approximately one fifth of the files that have post-release defects.

Having a small percentage of files does not necessarily mean less code, since some files are larger than others. Therefore, we also examine the amount of LOC that these files represent. Table 1 shows that on average, the amount of LOC that these breakage and surprise files make up is 3.2% and 3.8%, respectively. This is approximately one fourth the LOC of files with the post-release defects. The reduction is both promising, because it narrows down the set of files to be flagged for review, and challenging, because predicting such unusual files is much harder.

Table 1 also compares the percentages of MRs representing post-release, breakage and surprise defects. On average, the percentage of breakage and surprise MRs is much smaller than that of post-release MRs. Since we use the surprise threshold from the previous release to determine the surprise threshold, we are not able to calculate surprise defects for the first release (R1.1).

Breakage and surprise defects are difficult to pinpoint, since they only appear in 2% of the files.

3.4 Breakages vs Surprise Defects

As mentioned earlier, breakage defects are high severity defects that break existing functionality. They are a common concept in industry, and their impact is understood to be quite high. However, surprise defects are a concept that we defined based on our own industrial experience. In this section, we would like to learn more about the characteristics of surprise defects and verify our assumption that surprise defects highly impact the development organization, by addressing the following questions:

- Are surprise defects different than breakage defects? If so, what are the differences?
- Are surprise defects impactful?

Such verification helps us appreciate the value of studying surprise defects and to understand the implications of our findings for future research in the field.

Table 1: Percentage of Files, LOC and MRs containing Post-release, Breakage and Surprise defects.

Release	Post-Release			Breakage			Surprise		
	Files	**LOC**	**MRs**	**Files**	**LOC**	**MRs**	**Files**	**LOC**	**MRs**
R1.1	21.8	27	78.8	1.6	2.6	29.4	-	-	-
R2.1	6.5	8.4	48.6	2.1	2.6	27.1	0.2	0.4	1.5
R3.0	7.8	12.7	54.5	2.1	3.8	28.4	3.3	6.4	13.6
R4.0	11	18	79.7	1.4	1.3	25.6	3.0	5.2	11.5
R4.1	5.0	6.8	46.1	2.5	2.8	22.4	1.5	3.2	6.1
Average	**10.4**	**14.6**	**61.5**	**2.0**	**3.2**	**26.6**	**2.0**	**3.8**	**8.2**

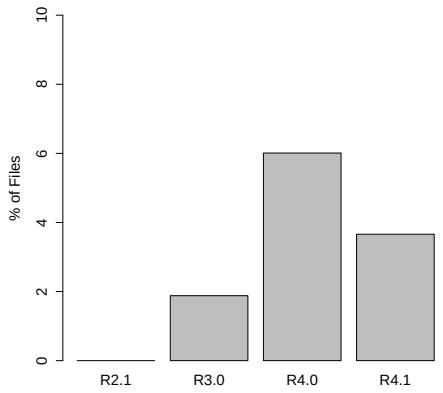

Figure 1: Percentage of Files Containing both, Surprise and Breakage Defects

Are surprise defects different than breakage defects? If so, what are the differences?

First, we look at the locations where breakage and surprise defects occur. If we, for example, find that breakage and surprise defects occur in the same location, then we can assume that the prediction models for breakage-prone files will suffice for surprise defect prediction.

To quantify the percentage of files that contain both types of defects, we divide the files into two sets: files with breakages and files with surprise defects. Then, we measure the intersection of the files in the two sets divided by the union of the files in the two sets:

$$\frac{Breakages \bigcap Surprise}{Breakages \bigcup Surprise}.$$

Figure 1 shows the percentage of files that have both breakages and surprise defects. At most 6% of the breakage and surprise files overlap. This low percentage of overlap shows that breakages and surprises are two very different types of defects. However, further examination of their specific characteristics is needed to better understand the potential overlap.

Therefore, we investigate the characteristics of breakage and surprise defects along the different defect prediction dimensions. For example, previous work showed that the amount of activity (i.e., number of pre-release changes) is a good indicator of defect-proneness, therefore, we look at the activity of breakage and surprise defects to compare and contrast. We summarize our findings, which are statistically significant with a p-value < 0.05, as follows:

Activity. Comparing the activity, measured in number of MRs, of breakage and surprise files to non-breakage and non-surprise files shows that breakage and surprise files have less activity. This could indicate that perhaps breakage and surprise files were inspected less.

MR size. Breakage and surprise files are modified by larger MRs. On average, a breakage MR touches 47 files as compared to 8 files touched for a non-breakage MRs. Surprise MRs touch 71 files on average as compared to 13 files touched by non-surprise MRs.

Time. The start times of MRs that touch breakage and surprise files show that for a particular release, breakage files are modified earlier than average, whereas surprise defect files are on average, worked on closer to the release date. This suggests that considerably less time was available for the development and testing of surprise files.

Functionality. By definition, breakage files are involved with legacy features (e.g., core features of the communication software), whereas surprise defect files are involved with more recent features (e.g., the porting of the studied software system to work in virtual environments).

Maintenance efforts. Breakage and surprise defect files are involved in more code addition MRs than the average non-breakage or non-surprise files. However, breakage files were involved in more fixes than surprise defect files.

Are surprise defects impactful?

The above comparisons show that breakage and surprise defects are different. To assess the impact of surprise defects we selected the five files with the highest surprise ratios in the last studied release of the software for an in-depth evaluation.

To evaluate the five files with the highest surprise score, we selected the latest changes prior to GA and the changes (defects) within one month after the GA for each of these files and investigated the nature and the origin of these defects by reading defect descriptions, resolution history and notes, inspection notes, as well as changes to the code. The five files were created eight, six, five, and, (for two files), one year(s) prior to GA. There were seven fixes within one month after the GA in these five files. Five defects had high severity and only two had medium severity, indicating that all of them were important defects. At least one fix in each of the five files was related to incomplete or inadequate requirements, i.e., introduced at the time of file creation or during the last enhancement of functionality.

For each post-GA defect in these five files, we traced back through changes to determine the first change that introduced the defect. Our intuition was that if the prediction can point to the change introducing the defect, it would limit the scope of the developers inspecting the flagged area to the kind of functionality changed and to the particular lines of code, thus simplifying the task of determining the nature of the potential defect or, perhaps, even highlighting the avenues for fixing it.

The results were surprising. In none of the cases the last pre-GA change could have been the cause of the defect. In the five defects related to inadequate requirements, we had to trace all the way to the initial implementation of the file. In the case of the two more recent files, at least we found a relationship between the pre-GA change (that was fixing inadequate requirements) and the post-GA fix that was fixing another shortcoming of the same requirements that was not addressed by the prior fix. For the two remaining defects introduced during coding or design phases, we had to trace back at least three changes to find the cause.

This qualitative investigation supports our intuition about the nature of surprise defects and our findings of the differences between breakages and surprise defects. The surprise defects appear to be an important kind of unexpected defect that appears to lurk in the code for long periods of time before surfacing. It also suggests a possible mechanism by which the prediction of surprise defects might work. As the usage profile or the intensity of usage changes over the years, early warnings are given by customers wanting to adjust a feature (one of the defects), or hard-to reproduce defects are starting to surface. Two of the defects appear to be caused by the increased use of multicore architecture that resulted in hard-to-reproduce timer errors and interactions with the system clock. At some point a major fault (surprise) is discovered (system restarting because of full message buffer, or because of data corruption) and the shortcomings of the requirements are finally addressed by the fix to the surprise defect.

Breakage and surprise defects are unique and different. Surprise defects also have high severity and appear to indicate problems in the requirements.

4. CASE STUDY SETUP

Now that we have analyzed the characteristics of breakage and surprise defects, we want to examine the effectiveness of predicting the locations of breakage and surprise defects. We address the following research questions:

RQ1. Can we effectively predict which files will have breakage/surprise defects?

RQ2. Which factors are important for the breakage/surprise defect prediction models?

RQ3. What effect does each factor have on the likelihood of finding a breakage/surprise defect in a file?

In this section, we outline and discuss the case study setup. First, we present the various factors used to predict breakage and surprise defects. Then, we outline the prediction modeling technique used. Finally, we outline how we evaluate the performance of our prediction models.

4.1 Factors Used to Predict Breakage and Surprise Defects

The factors that we use to predict files with breakage or surprise defects belong to three different categories: 1) traditional factors found in previous defect prediction work, 2) factors associated to co-changed files and 3) time-related factors (i.e., the age of a file and the time since the last change to the file).

These factors are based on previous post-release defect prediction work and on our findings in Section 3.4.

Traditional Factors: Previous work shows that the number of previous changes, the number of pre-release defects and the size of a file are good indicators of post-release defects [11, 35]. Since breakage and surprise defects are a special case of post-release defects, we believe that they can help us predict breakage and surprise defects as well.

Co-change Factors: By definition, breakage and surprise defects are not expected to happen. One possible reason for their occurrence could be hidden dependencies (e.g., logical coupling [9]). The intuition here is that a change to a co-changed file may cause a defect. Since more recent changes are more relevant, we also calculate for each factor its value in the three months prior to release, labeled as "recent".

Time Factors: To take into account the fact that breakage and surprise defects start earlier and later than average, respectively, we consider factors related to how close to a release a file is changed, as well as the file's age.

Each category comprises several factors, as listed in Table 2. For each factor, we provide a description, motivation and any related work.

4.2 Prediction Models

In this work, we are interested in predicting whether or not a file has a breakage or surprise defect. Similar to previous work on defect prediction [35], we use a logistic regression model. A logistic regression model correlates the independent variables in Table 2) with the dependent variable (probability of the file containing a breakage or surprise defect).

Initially, we built the logistic regression model using all of the factors. However, to avoid collinearity problems and to assure that all of the variables in the model are statistically significant, we removed highly correlated variables (i.e., any variables with correlation higher than 0.5). We performed this removal in an iterative manner, where we measured the correlation of a factor with all other factors and kept the factor that had the most factors correlated with it. We repeated this process until all the factors in the left in the model had a Variance Inflation Factor (VIF) below 2.5, as recommended by previous work [4]. This way, we are left with the least number of uncorrelated factors in the final model. To test for statistical significance, we measure the p-value of the independent variables in the model to make sure that this is less than 0.1.

Altogether, we extracted a total of 15 factors. After removing the highly correlated variables, nine factors were left that covered all three categories. It is important to note however that each release had a different set of factors.

4.3 Performance Evaluation of the Prediction Models

After building the logistic regression model, we verify its performance using two criteria: Explanatory Power and Predictive Power. These measures are widely used to measure the performance of logistic regression models in defect prediction [6, 35].

Explanatory Power. Ranges between 0-100%, and quantifies the variability in the data explained by the model. We also report and compare the variability explained by each independent variable in the model. Examining the explained variability of each independent variable allows us to quantify the relative importance of the independent variables in the model.

Predictive Power. Measures the accuracy of the model in predicting the files that have one or more breakage/surprise defects. The accuracy measures that we use (precision and recall) are based on the classification results in the confusion matrix (shown in Table 3).

1. **Precision:** the percentage of correctly classified breakage/surprise files over all of the files classified as having breakage/surprise defects: Precision $= \frac{TP}{TP+FP}$.

Table 2: List of Factors Used to Predict Breakage and Surprise Defects.

Category	Factor	Description	Rationale	Related Work
Traditional Factors	pre_defects	Number of pre-release defects	Traditionally performs well for post-release defect prediction.	Prior defects are a good indicator of future defects [33].
	pre_changes	Number of pre-release changes	Traditionally performs well for post-release defect prediction.	The number of prior modifications to a file is a good predictor of future defects [1, 11, 18].
	file_size	Total number of lines in the file	Traditionally performs well for post-release defect prediction.	The lines of code metric correlates well with most complexity metrics (e.g., McCabe complexity) [11, 15, 18, 27].
Co-change Factors	(recent) num_co-changed_files	Number of files a file co-changed with	The higher the number of files a file co-changes with, the higher the chance of missing to propagate a change.	The number of files touched by a change is a good indicator of its risk to introduce a defect [22]. We apply this factor to the number of files co-changing with a file.
	(recent) size_co-changed_files	Cumulative size of the co-changed files	The larger the co-changed files are, the harder they are to maintain and understand.	The simple lines of code metric correlates well with most complexity metrics (e.g., McCabe complexity) [11, 15, 18, 27]. We apply this factor to co-changing files.
	(recent) modification_ size_co-changed_files	The number of lines changed in the co-changed files	The larger the changes are to the co-changed files, the larger the chance of introducing a defect.	Larger changes have a higher risk of introducing a defect [22]. We apply this factor to a file's co-changing files.
	(recent) num_changes_ co-changed_files	Number of changes to the co-changed files	The higher the number of changes to the co-changed file, the higher the chance of introducing defects.	The number of prior modifications to a file is a good predictor of its future defects [1, 11, 18]. We apply this factor to a file's co-changing files.
	(recent) pre_defects_ co-changed_files	Number of pre-release defects in co-changed files	The higher the number of pre-release defects in the co-changed files, the higher the chance of a defect.	Prior defects are a good indicator of future defects [33]. We apply this factor to a file's co-changing files.
Time Factors	latest_change_ before_release	The time from the latest change to the release (in days)	Changes made close to the release date do not have time to get tested properly.	More recent changes contribute more defects than older changes [11].
	age	The age of the file (from first change until the release GA date)	The older the file, the harder it becomes to change and maintain.	Code becomes harder to change over time [8].

Table 3: Confusion Matrix

	True class	
Classified as	Breakage	No Breakage
Breakage	TP	FP
No Breakage	FN	TN

2. **Recall:** the percentage of correctly classified breakage/surprise files relative to all of the files that actually have breakage/surprise defects: Recall $= \frac{TP}{TP+FN}$.

A precision value of 100% would indicate that every file we classify as having a breakage/surprise defect, actually has a breakage/surprise defect. A recall value of 100% would indicate that every file that actually has a breakage was classified as having a breakage/surprise.

We employ 10-fold cross-validation [7]. The data set is divided into two parts, a testing data set that contains 10% of the original data set and a training data set that contains 90% of the original data set. The model is trained using the training data and its accuracy is tested using the testing data. We repeat the 10-fold cross validation 10 times by randomly changing the fold. We report the average of the 10 runs.

Determining the logistic regression model threshold value: The output of a logistic regression model is a probability (between 0 and 1) of the likelihood that a file belongs to the true class (e.g., a file is buggy). Then, it is up to the user of the output of the logistic regression model to determine a threshold at which she/he will consider a file as belonging to the true class. Generally speaking, a threshold of 0.5 is used. For example, if a file has a likelihood of 0.5 or higher, then it is considered buggy, otherwise it is not.

However, the threshold is different for different data sets and the

value of the threshold affects the precision and recall values of the prediction models. In this paper, we determine the threshold for each model using an approach that examines the tradeoff between type I and type II errors [22]. Type I errors are files that are identified as belonging to the true class, while they are not. Having a low logistic regression threshold (e.g., 0.01) increases type I errors: a higher fraction of identified files will not belong to the true class. A high type I error leads to a waste of resources since many non-faulty files may be wrongly classified. On the other hand, type II error is the fraction of files in the true class that are not identified as being true when they should be. Having a high threshold can lead to large type II errors, and thus missing many files that may be defective.

To determine the optimal threshold for our models, we perform a cost-benefit analysis between the type I and type II errors. Similar to previous work [22], we vary the threshold value between 0 to 1 and use the threshold where the type I and type II errors are equal.

4.4 Measuring the Effect of Factors on the Predicted Probability

In addition to evaluating the accuracy and explanatory power of our prediction models, we need to understand the effect of a factor on the likelihood of finding a breakage or surprise defect. Quantifying this effect helps practitioners gain an in-depth understanding of how the various factors relate to breakage and surprise defects.

To quantify this effect, we set all of the factors to their median value and record the predicted probabilities, which we call the Standard Median Model (SMM). Then, to measure the individual effect of each factor, we set all of the factors to their median value, except for the factor whose effect we want to measure. We double the median value of that factor and re-calculate the predicted values, which we call the Doubled Median Model (DMM).

We then subtract the predicted probability of the SMM from the predicted output of the DMM and divide by the predicted probability of the SMM. Doing so provides us with a way to quantify the effect a factor has on the likelihood of a file containing a breakage or surprise defect.

The effect of a factor can be positive or negative. A positive effect means that a higher value of the factor increases the likelihood, whereas a negative effect value means that a higher value of the factor decreases the likelihood of a file containing a breakage/surprise defect.

5. CASE STUDY RESULTS

In this section, we address the research questions posted earlier. First, we examine the accuracy (in terms of predictive and explanatory power) of our prediction models. Then, we examine the contribution of each factor on the models in terms of explanatory power. Lastly, we examine the effect of the factors on the breakage- and surprise-proneness.

RQ1. Can we effectively predict which files will have breakage/surprise defects?

Using the extracted factors, we build logistic regression models that aim to predict whether or not a file will have a breakage or surprise defect. The prediction was performed for five different releases of the large commercial software system. To measure predictive power, we present the precision, recall and the threshold (Th.) value used in the logistic regression model, for each release. The last row in the tables presents the average across all releases.

Predictive power: Tables 4 and 5 show the results of the breakage and surprise defect prediction models, respectively. On average, the precision for breakage and surprise defects is low, i.e., 4.4% for breakages and 5.9% for surprise defects.

It is important to note that the low precision value is due to the low percentage of breakage and surprise defects in the data set (as shown earlier in Table 1). As noted by Menzies *et al.* [21], in cases where the number of instances of an occurrence is so low (i.e., 2%), achieving a high precision is extremely difficult, yet not that important. In fact, a random prediction would be correct 2.0% of the time, on average, whereas our prediction model more than doubles that precision for breakages and approximately triples that for surprise defects. The more important measure of the prediction model's performance is recall [21], which, on average is 69.0% for breakage defects and 74.0% for surprise defects.

Explanatory power: The explanatory power of the models ranges between 6.85 - 16.96% (average of 12.35%) for breakage defects and between 4.1 - 30.64% (average of 17.26%) for surprise defects. The values that we achieve here are comparable to those achieved in previous work predicting post-release defects [4, 30].

The explanatory power may be improved if more (or better) factors are used in the prediction model. We view the factors used in our study as a starting point for breakage and surprise defect prediction and plan to (and encourage others to) further investigate in order to improve the explanatory power of these models.

Other considerations: For each prediction model, we explicitly report the threshold for the logistic regression models as Th. (pred) in Tables 4 and 5. In addition, the surprise threshold, which we use to identify files that had surprise defects, is given under the Th. (sup) column in Table 5.

Since the number of pre-release defects is used in the dependent variable of the surprise model ($Surprise(file) = \frac{post\ defects}{pre\ defects}$), we did not use it as part of the independent variables in the prediction model. This makes our results even more significant, since previous work [23, 35] showed that pre-release defects traditionally are the largest contributor to post-release defect prediction models.

Table 4: Performance of Breakage Prediction Models

	Predictive Power			Explanatory Power
Release	Precision	Recall	Th. (pred)	Deviance Explained
R1.1	3.6	69.0	0.46	16.96%
R2.1	4.3	64.3	0.47	6.85%
R3	4.8	69.6	0.49	14.49%
R4	4.1	73.8	0.49	12.09%
R4.1	5.4	68.5	0.46	11.38%
Average	4.4	69.0	0.47	12.35%

Table 5: Performance of Surprise Prediction Models

	Predictive Power				Explanatory Power
Release	Precision	Recall	Th. (pred)	Th. (sup)	Deviance Explained
R1.1	-	-	-	-	-
R2.1	1.4	75.0	0.56	2.4	4.10 %
R3	7.3	69.0	0.44	1.7	10.79 %
R4	9.8	75.8	0.38	1.6	30.64 %
R4.1	4.9	75.4	0.34	1.5	23.52 %
Average	5.9	74.0	0.43	1.8	17.26 %

Our prediction models predict breakage and surprise defects with a precision that is at least double that of a random prediction and a recall above 69%.

RQ2. Which factors are important for the breakage/surprise defect prediction models?

In addition to knowing the predictive and explanatory power of our models, we would like to know which factors contribute the most to these predictions. To answer this question, we perform an ANOVA analysis to determine the contribution of the three factor categories. We summarize the findings in Tables 6 and 7 as follows:

Traditional Defect Prediction Factors: are major contributors for the breakage defect prediction models, however, they only have a small contribution in predicting surprise defects.

Co-Change Factors provide a small contribution to breakage defect prediction models, however, they make a major contribution to predicting surprise defects.

Time Factors provide a minor contribution to breakage defect prediction models, however, they make a major contribution in predicting surprise defects.

Although there are exceptions to the above mentioned observations (e.g., traditional defect prediction factors make a major contribution to the surprise defect model in R2.1), our observations are based on the trends observed in the majority of the releases.

As mentioned earlier in Section 3.4, breakage files were mainly involved with defect fixing efforts and are, by definition, defects that are found in the field. Perhaps these characteristics of breakage defects help explain why the traditional post-release defect prediction factors perform well in predicting breakages.

Furthermore, earlier observations in Section 3.4 showed that surprise defect files were worked on later than other files (i.e., MRs that touched surprise defect files were started later than other MRs). Our findings show that being changed close to a release is one of the best indicators of whether or not a file will have a surprise defect.

Traditional defect prediction factors are good indicators of breakage defects. The factors related to co-changed files and time-related factors are good indicators of surprise defects.

RQ3. What effect does each factor have on the likelihood of finding a breakage/surprise defect in a file?

Thus far we examined the prediction accuracy and the importance of the factors to these prediction models. Now, we study the effect of each factor on the likelihood of finding a breakage or surprise defect in a file. In addition to measuring the effect, we also consider stability and explanatory impact. If an effect has the same sign/direction for all releases (i.e., positive or negative effect in all releases), then we label it as highly stable. If the effect of a factor has the same sign in all releases except for one, then we label it as being mainly stable. A factor having a different sign in more than two of the five releases is labeled as being unstable. The explanatory impact column is derived from the values in Tables 6 and 7. If a factor belongs to a category that had a strong explanatory power, then we label it as having high impact, otherwise we consider it as having low impact.

We use the stability and impact measure to help explain the strength of our findings. For example, if we find that a factor has a positive effect and has high impact to the explanatory power of the model, then we believe this effect to be strong.

Breakage Defects: Table 8 shows the effect of the different factors on the likelihood of predicting a breakage defect in a file. We observe that in all releases, pre-release defects have a positive effect on the likelihood of a breakage defect existing in a file (i.e., highly stable). In addition, Table 6 showed that the traditional defect prediction factors contributed the most to the explanatory power of the model (i.e., high impact). File size generally has a positive effect. The latest change before release factor had low impact that is non-stable.

As stated earlier, our manual examination of the breakage files showed that breakage files were especially involved with fixing efforts. Therefore, the fact that pre-release defects and size have a strong positive effect was expected (since these factors are positively correlated with post-release defects). In fact, we found that the average file size of breakage files is 50% larger than non-breakage files. The rest of the factors had low impact on the explanatory power of the prediction model, therefore we cannot conclude any meaningful results from their effect.

Release 4 (R4) in our results seems to be an outlier. For example, contrary to the other releases, file size shows a negative effect in R4. After closer examination, we found that R4 was a large major release that added a large amount of new functionality. This could be the reason why the effect values for R4 are so different from the effect results of the remaining releases.

Surprise Defects: Table 9 shows the effect values for the surprise defect prediction model. In this model, file size has a large stable positive effect, however as shown earlier in Table 7 this factor category has very little contribution to the explanatory power of the model (i.e., low impact).

We find that making a change last minute increases the likelihood of a surprise defect (i.e., the negative effect of latest change before release). As shown in Section 3.4, in contrast to breakages, surprise defect files were worked on later than usual. We conjecture that starting late means less time for testing, hence the much higher effect of these late changes on surprise defect files compared to the breakage files.

Pre-release defects and file size have a positive effect on the likelihood of a file containing a breakage defect. The time since last change before release has a negative effect on the likelihood of a file having a surprise defect.

6. EFFORT SAVINGS BY FOCUSING ON SURPRISE AND BREAKAGE FILES

Thus far, we have shown that we are able to build prediction models to predict files that contain breakage and surprise defects. However, one question still lingers: what if we used the traditional post-release defect prediction model to predict breakage and surprise defects? Is it really worth the effort to build these specialized models?

To investigate whether building specialized prediction models for breakage and surprise defects is beneficial, we use defect prediction models that are trained to predict post-release defects, to predict files with breakages and surprise defects. Due to the fact that post-release defects are much more common than breakages or surprise defects, post-release defect models are more likely to say that most files have breakages or surprise defects. That will lead to

Table 6: Contribution of Factor Categories to the Explanatory Power of Breakage Prediction Models

	R1.1	R2.1	R3	R4	R4.1
Traditional Defect Prediction Factors	15.60%	6.80%	11.16%	8.92%	10.75%
Co-Change Factors	0.59%	0.03%	2.93%	1.07%	0.30%
Time Factors	0.77%	0.01%	0.39%	2.09%	0.32%
Overall Deviance Explained	**16.96%**	**6.85%**	**14.48%**	**12.09%**	**11.38%**

Table 7: Contribution of Factor Categories to the Explanatory Power of Surprise Prediction Models

	R1.1	R2.1	R3	R4	R4.1
Traditional Defect Prediction Factors	-	2.94%	2.76%	0.68%	0.69%
Co-Change Factors	-	0.29%	4.56%	9.55%	1.52%
Time Factors	-	0.86%	3.46%	20.39%	21.30%
Overall Deviance Explained	-	**4.10%**	**10.79%**	**30.64%**	**23.52%**

Table 8: Effect of factors on the likelihood of predicting a file with a breakage defect. Effect is measured by setting a factor to double its median value (1 if the median is 0), while the rest of the factors are set to their median value.

	R1.1	R2.1	R3	R4	R4.1	Stability	Explanatory Impact
pre_defects	124%	105%	85%	68%	121%	Highly Stable	High Impact
file_size	263%	120%	575%	-51%	18%	Mainly Stable	High Impact
modification_size_co_changed_files	744%	-	-	-	-	-	Low Impact
num_co_changed_files	-71%	-	-	887%	-	-	Low Impact
num_co_changed_files_per_mr	-	-	-91%	-	-	-	Low Impact
recent_num_co_changed_files	-	-	-	-6%	-	-	Low Impact
recent_modification_size_co_changed_files	2%		13%	-	2%	-	Low Impact
recent_modification_size_co_changed_files_per_mr	-	1%	-	-	-	-	Low Impact
latest_change_before_release	-58%	-16%	85%	-88%	-55%	Not Stable	Low Impact

Table 9: Effect of factors on the likelihood of predicting a file with a surprise defect. Effect is measured by setting a factor to double its median value (1 if the median is 0), while the rest of the factors are set to their median value.

	R1.1	R2.1	R3	R4	R4.1	Stability	Explanatory Impact
file_size	-	1417%	260%	184%	128%	Highly Stable	Low Impact
num_co_changed_files	-	-	-	-67%	-	-	High Impact
num_co_changed_files_per_mr	-	-	-75%	-	-	-	High Impact
recent_num_co_changed_files	-	-	-	-44%	-	-	High Impact
recent_modification_size_co_changed_files	-	-	-3%	-	-20%	-	High Impact
recent_modification_size_co_changed_files_per_mr	-	5%	-	-	-	-	High Impact
latest_change_before_release	-	-63%	-65%	-97%	-96%	Highly Stable	High Impact

a large amount of unnecessary work. Therefore, we use the number of false negatives to compare the performance of the post-release models and the specialized models. To make a meaningful comparison of effort (which is related to Type I error), we fix Type II error to be the same in both models.

Breakage Defects. Table 10 shows the results of the specialized prediction model (Breakage -> Breakage) and the post-release prediction model (Post -> Breakage) for release 4.1. Both of these models predict files that have breakage defects. The false positives are highlighted (in grey) in Table 10. We observe that the specialized prediction model has approximately 3.3% (i.e., $\frac{688-665}{688}$) less false positives than the post-release model. This means that using the specialized model would reduce the inspection effort of files by 3.3%. We also convert this effort saving into LOC, which is approximately 24.3% of the total LOC.

Surprise Defects. Table 11 shows the results of the specialized model (Surprise -> Surprise) and the post-release prediction model (Post -> Surprise) for files that have surprise defects. In this case, the specialized models lead to approximately 30% (i.e., $\frac{673-471}{673}$) effort savings (i.e., less false positives). Comparing the savings in terms of LOC, we find that using the specialized prediction model leads to approximately 21.2% effort savings compared to using a

Table 10: Breakages in Release 4.1

	Breakage -> Breakage Predicted			Post -> Breakage Predicted	
Actual	0	1	Actual	0	1
0	748	665	0	725	688
1	7	26	1	7	26

Table 11: Surprise in R4.1

	Surprise -> Surprise Predicted			Post -> Surprise Predicted	
Actual	0	1	Actual	0	1
0	957	471	0	755	673
1	4	14	1	4	14

traditional post-release defect prediction model. This is a considerable amount of effort savings and shows the benefits of building a specialized prediction model of files with surprise defects.

Using our custom prediction models reduces the amount of files inspected by practitioners by 3.3% for breakages and 30% for surprise defects.

7. LIMITATIONS

Threats to Construct Validity consider the relationship between theory and observation, in case the measured variables do not measure the actual factors.

Breakage MRs were manually identified in our data by project experts. Although this manual linking was done by these project experts, some MRs may have been missed or incorrectly linked.

We used files in defects reported within one month after release to determine the surprise defect threshold. The assumption here is that defects reported within one month involve important functionality that is widely used after release. Defects that affect important functionality may be reported later than one month, however.

Threats to External Validity consider the generalization of our findings. The studied project was a commercial project written mainly in C/C++, therefore, our results may not generalize to other commercial or open source projects.

8. LESSONS LEARNED AND FUTURE WORK

After performing our study, we asked the opinions of the highly experienced quality manager in the project about the prediction results. The manager has a theory about the reported effect of our $last\_change\_before\_release$ factor. The theory is that the so called "late fix frenzies" that go on in organizations to bring down the number of open defects in a software system before the release, might have compromised the quality of inspections and other quality assurance activities. This suggests that prediction may help to quantify and confirm intuition about the relationships between the aspects of the development process and the high-impact defects.

However, when the quality manager considered the merits of our prediction by their utility for system verification she argued that identifying locations of defects is of limited use to system testers because they test system features or behaviors, not individual files. In addition, she doubted that the predicted location could be helpful even to developers doing inspections or unit tests without the additional information about the nature of the problem or how it should be fixed.

Despite the positive findings related to prediction quality, narrowing the scope, and effort savings, we still appear to be far from the state where the prediction results could be used in practice. Based on our quantitative and qualitative findings and experience we hypothesize that for defect prediction to become a practical tool, each predicted location has to also contain a clear suggestion on why the defect might be there and how it may be fixed.

To achieve this, we propose a procedure similar to the one we conducted to identify surprise defects, to classify defects into a variety of classes according to their nature, the ways they may have been introduced, and to the ways they may need to be fixed. We expect that each type of high-impact defect would have a different prediction signature, which, in turn, can be used to provide developers with a recommendation on where the defect may be, what nature it may have, and how it may be fixed. We can see an example of such a classification in the static analysis tools that not only provide a warning, but also give a reason why a particular pattern might be a defect and a clear suggestion on how the potential defect can be fixed. We are not aware of any similar patterns for defect prediction.

For example, our investigation of surprise defects could be used to provide a warning of the kind: "This file might contain a defect that has been introduced a while ago, perhaps because of incorrect requirements. Change patterns to this file suggest that the usage profile might have changed recently and the requirements may need to be reviewed to make sure they are accurate and complete." Obviously, a more extensive investigation may be needed to provide more specific recommendations and we believe that the defect prediction methods should be tailored not simply to predict defect locations, but, like basic static analysis tools such as `lint`, should also detect patterns of changes that are suggestive of a particular type of defect, and recommend appropriate remedies.

9. CONCLUSION

The majority of defect prediction work focuses on predicting post-release defects, yet the adoption of this work in practice remains relatively low [10, 13]. One of the main reasons for this is that defect prediction techniques generally identify too many files as having post-release defects, requiring significant inspection effort from developers.

Instead of considering all defects as equal, this paper focuses on predicting a small subset of defects that are highly impacting. Since there are many different interpretations of "high impact", we focus on one interpretation from the perspective of customers (breakage defects) and one from the perspective of practitioners (surprise defects). We find that:

- Both kinds of defects are different and that surprise defects, similar to the more established concept of breakage defects, have a high impact.
- Specialized defect prediction models can predict breakage and surprise defects effectively, yielding sizeable effort savings over using simple post-release defect prediction models.
- Traditional defect prediction factors (i.e., the number of pre-release defects and file size) are good predictors of breakage defects, whereas the number of co-changed files, the size of recently co-changed files and the time since the last change are good predictors of surprise defects.

Our findings suggest that building specialized prediction models is valuable to bring defect prediction techniques closer to adoption in practice.

However, our qualitative analysis of surprise defects and the feedback from the quality assurance manager clearly indicate that further work is needed to develop defect prediction into a practical tool. In particular, we found support for the idea of building specialized models that identify not only a defect's location, but also its nature, thus greatly simplifying the process of determining what the defect is and how it needs to be fixed.

10. REFERENCES

[1] E. Arisholm and L. C. Briand. Predicting fault-prone components in a java legacy system. In *Proc. Int'l Symposium on Empirical Softw. Eng. (ISESE'06)*, pages 8–17, 2006.

[2] E. Arisholm, L. C. Briand, and E. B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. Syst. Softw.*, 83(1):2–17, 2010.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[4] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Trans. Softw. Eng.*, 99(6):864–878, 2009.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[6] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Proc. Int'l Working Conf. on Mining Software Repositories (MSR'10)*, pages 31–41, 2010.

[7] B. Efron. Estimating the error rate of a prediction rule: Improvement on Cross-Validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.

[8] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE Trans. Softw. Eng.*, 27:1–12, 2001.

[9] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proc. Int'l Conf. on Software Maintenance (ICSM'98)*, pages 190–198, 1998.

[10] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *IEEE Software*, 26(1):67 –70, 2009.

[11] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7):653–661, 2000.

[12] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Softw. Eng.*, 31(10):897–910, 2005.

[13] A. Hassan. The road ahead for mining software repositories. In *Frontiers of Software Maintenance (FoSM'08)*, pages 48–57, 2008.

[14] A. E. Hassan. Predicting faults using the complexity of code changes. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'09)*, pages 78–88, 2009.

[15] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles. Towards a theoretical model for software growth. In *Proc. Int'l Workshop on Mining Software Repositories (MSR'07)*, pages 21–29, 2007.

[16] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan. Revisiting common bug prediction findings using effort aware models. In *Proc. Int'l Conf. on Software Maintenance (ICSM'10)*, pages 1–10.

[17] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl. Data mining for predictors of software quality. *International Journal of Software Engineering and Knowledge Engineering*, 9(5):547–564, 1999.

[18] M. Leszak, D. E. Perry, and D. Stoll. Classification and evaluation of defects in a project retrospective. *J. Syst. Softw.*, 61(3), 2002.

[19] T. J. McCabe. A complexity measure. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'76)*, page 407, 1976.

[20] T. Mende and R. Koschke. Revisiting the evaluation of defect prediction models. In *Proc. Int'l Conf. on Predictor Models in Software Engineering (PROMISE'09)*, pages 7:1–7:10, 2009.

[21] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald. Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'". *IEEE Trans. Softw. Eng.*, 33:637–640, September 2007.

[22] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[23] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'08)*, pages 181–190, 2008.

[24] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'05)*, pages 284–292, 2005.

[25] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. Int'l Conf. on Softw. Eng. (ICSE'06)*, pages 452–461, 2006.

[26] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Trans. Softw. Eng.*, 22(12):886–894, 1996.

[27] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Where the bugs are. In *Proc. Int'l Symposium on Software Testing and Analysis (ISSTA'04)*.

[28] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, 2005.

[29] E. Shihab. Pragmatic prioritization of software quality assurance efforts. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 1106–1109, 2011.

[30] E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan. Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project. In *Proc. Int'l Symposium on Empirical Softw. Eng. and Measurement (ESEM'10)*, pages 1–10, 2010.

[31] Y. Shin, A. Meneely, L. Williams, and J. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Trans. Softw. Eng.*, PP(99):1, 2010.

[32] R. Subramanyam and M. S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. Softw. Eng.*, 29(4):297–310, 2003.

[33] T.-J. Yu, V. Y. Shen, and H. E. Dunsmore. An analysis of several software defect models. *IEEE Trans. Softw. Eng.*, 14(9), 1988.

[34] T. Zimmermann, N. Nagappan, and L. Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *Proc. Int'l Conf. on Software Testing, Verification and Validation (ICST'10)*, pages 421–428, 2010.

[35] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proc. Int'l Workshop on Predictor Models in Software Engineering (PROMISE'07)*, pages 1–7, 2007.