Are Change Metrics Good Predictors for an Evolving Software Product Line?

Sandeep Krishnan
Dept. of Computer Science
Iowa State University
Ames, IA 50014
sandeepk@iastate.edu

Chris Strasburg
Iowa State University
& Ames Laboratory, US DOE
cstras@iastate.edu

Robyn R. Lutz
Dept. of Computer Science
Iowa State University
& JPL/Caltech
rlutz@iastate.edu

Katerina Goševa-Popstojanova
Lane Dept. of CSEE
West Virginia University
Morgantown, WV 26506-6109
Katerina.Goseva@mail.wvu.edu

ABSTRACT

Background: Previous research on three years of early data for an Eclipse product identified some predictors of failure-prone files that work well for that data set. Additionally, Eclipse has been used to explore characteristics of product line software in previous research.
Aims: To assess whether change metrics are good predictors of failure-prone files over time for the family of products in the evolving Eclipse product line.
Method: We repeat, to the extent possible, the decision tree portion of the prior study to assess our ability to replicate the method, and then extend it by including four more recent years of data. We compare the most prominent predictors with the previous study's results. We then look at the data for three additional Eclipse products as they evolved over time. We explore whether the set of good predictors change over time for one product and whether the set differs among products.
Results: We find that change metrics are consistently good and incrementally better predictors across the evolving products in Eclipse. There is also some consistency regarding which change metrics are the best predictors.
Conclusion: Change metrics are good predictors for failure-prone files for the Eclipse product line. A small subset of these change metrics is fairly stable and consistent across products and releases.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Product metrics, Process metrics*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PROMISE '11, September 20–21, 2011, Banff, Canada
Copyright 2011 ACM 978-1-4503-0709-3/11/09 ...$10.00.

General Terms

Reliability

Keywords

Software product lines, change metrics, reuse, prediction, post-release defects, failure-prone files

Project: Eclipse (eclipse.org)
Content: Defect counts (non-trivial post-release)
Releases: 2.0, 2.1, 3.0, 3.3, 3.4, 3.5, and 3.6
Level: Files
URL: `http://www.cs.iastate.edu/~lss/PROMISE11Data.tar.gz`

Figure 1: Data set summary

1. INTRODUCTION

What is unique about software product lines as distinct from other systems is that there is a high degree of commonality among all the systems in the product line but they may differ one from another via a set of allowed variations. Commonalities are implemented in files that are reused in every product. Variations are implemented in files that are available for reuse in a subset of products requiring the same options or alternatives.

Ongoing change is typical in product lines and proceeds along two main dimensions. The first dimension is evolution of the product line in which, as the product line matures, more products are built. Some of these additional products may include new features (i.e., units of functionality [3]). The changes also may propagate to other, previously built products [35]. If the changes are incorporated into the product line, the product line asset repository is updated so that future products can reuse them.

The second dimension of product line evolution is changes in an individual product from one of its releases to another. This is similar to the evolution and maintenance of a single system, except that it may happen to each system in the product line. In previous work we have shown that even files implementing commonalities experience change on an on-going basis [16].

This high degree of commonality and low degree of variations mean that we may well be able to learn something about predicting failure-prone files in the product line from information about changes, bug fixes, and failures experienced by previous systems in the product line. A failure-prone file is a file with one or more non-trivial post-release bugs recorded in the Eclipse Bugzilla database.

The work presented in this paper is motivated by the following research questions. Are there any change metrics that serve as good predictors for which files are failure prone as a product matures over releases? Do any of these change metrics also serve as good predictors across all the products in a product line over time? Does our ability to predict the failure-prone files improve over time across products as the product line matures?

To investigate these questions, we explore here whether accurate and meaningful predictions of failure-prone files can be made, both across the sequential releases of a single product and across the various products in a product line. We study whether there are good predictors of failure-prone files for individual products in the product line, whether there are good predictors across the product line, and what the relationship is between them.

Our data-driven investigation is based on using the large open-source project Eclipse which, following Chastek, McGregor and Northrop [6], we consider as a product line. We build on previous work by Zimmermann, Premraj and Zeller [39] and by Moser, Pedrycz and Succi [25]. The authors in [39], a 2007 PROMISE paper, studied defects from the bug database of three early releases of an Eclipse product at both the file and package level. They annotated the data with code metrics, made it available, and built logistic regression models to predict post-release defects. At the file level, the models had mixed results, with low recall values less than 0.400 and precision values mostly above 0.600. The authors in [25], a 2008 paper, found that change metrics performed better than code metrics on a selected subset of the same Eclipse data set, and that the performance of the J48 decision tree learner surpassed the performance of logistic regression and Naïve Bayes learners.

We are most interested in observing the performance of the J48 machine learner on Eclipse and determining whether the set of prominent predictors changes both between products and as the product line evolves over time. Thus, in the work described in this paper, we first repeat the decision tree portion of the study presented in [25] to assess our ability to replicate the method, and then extend it by including four more recent years of data. For this work we use the J48 tree-based learning algorithm as implemented in Weka [12], as we are interested in both the prediction results and the tree structure [32]. We look at the evolution of one particular product, Eclipse Classic, over a period of 9 years. We observe the classification results during its early evolution (versions 2.0, 2.1, and 3.0), as in [25], but also look at its more recent evolution (the versions for years 2007, 2008, 2009, and 2010). We find some overlaps and some differences between the most prominent predictors over the shorter and longer time periods for these components. Performance of the J48 learner on our larger data set is similar or better than for the smaller data set in [25].

We then repeat the effort for three additional products in the Eclipse product line, Eclipse Java, Eclipse JavaEE and Eclipse C/C++, across the last four years of their evolution. Results show that a small set of change metrics are consistently good predictors and that prediction results of post-release failure-prone files improve over time in the Eclipse product line.

Several interesting findings resulting from the investigation are described in the rest of the paper. The main contributions of the work are:

- *Change metrics.* The change metrics provide good classification of the failure-prone files in the Eclipse product line.
- *Product evolution.* As each product evolves, there is a stable set of change metrics that are prominent predictors of failure-prone files across its releases.
- *Product line evolution.* There is some consistency among the prominent predictors for early vs. late releases for all the considered products in the product line. There is a subset of change metrics (number of revisions, maximum changeset and number of authors) that is among the prominent predictors of all the products across most of the releases.
- *Prediction trends.* As the product line matures, the learner performance improves (i.e., higher percentage of correctly classified files, higher true positive rate, and lower false positive rate) for each of the four Eclipse products.

The rest of the paper is organized as follows. Section 2 describes Eclipse and gives the reasons for considering it as a software product line. Section 3 presents the approach to data collection and analysis. Section 4 describes the observations and findings for evolution of single products. Section 5 describes the observations and findings for evolution of the product line. Section 6 summarizes and discusses the results in the context of software product lines. Section 7 considers threats to validity. Section 8 describes additional related work. Section 9 provides concluding remarks.

2. ECLIPSE PRODUCT LINE

A product line is "a family of products designed to take advantage of their common aspects and predicted variabilities" [36]. The systematic reuse and maintenance of code and other artifacts in the product line repository has been shown to support faster development of new products and lower-cost maintenance of existing products in many industries [2], [10], [30], [36]. As the common and variation code files are reused across products, they go through iterative cycles of testing, operation and maintenance that over time identify and remove many of the bugs that can lead to failures. There is thus some reason to anticipate that the quality and reliability of both the existing products and the new products may improve over time.

The lack of available product line data, however, makes it hard to investigate such and similar claims. The availability of Eclipse data is a noteworthy exception. The Eclipse project, described on its website as an ecosystem, documents and makes available bug reports, change reports, and source code that span the evolution of the Eclipse products.

Chastek, McGregor and Northrop [6] were the first that we know of to consider Eclipse from a product line perspective. Eclipse provides a set of different products tailored to

the needs of different user communities. Each product has a set of common features, yet each product differs from other products based on some variation features. The features are developed in a systematic manner with planned reuse for the future. The features are implemented in Eclipse as plug-ins and integrated to form products. The products in the Eclipse product line are thus the multiple package distributions provided by Eclipse for different user communities.

In previous work [16], we studied the failure and change trends at the component level for the commonalities and variations in the Eclipse product line. We found that as the product line evolves, fewer serious failures occur in components implementing commonalities than in components implementing variations, and that the common components also exhibit less change than the variable components over time. This motivated our current effort to understand whether the stabilizing behavior of the commonalities as the product line evolves supports prediction of failure-prone files over the product line.

3. APPROACH

3.1 Data Collection and Integration

In order to both replicate and extend the work conducted by Moser et al. [25], we collected Eclipse CVS log data and bug tracking database entries from May 2001 to the present. This data was partitioned into time periods corresponding with 6 months before and after the release of Eclipse 2.0, Eclipse 2.1, Eclipse 3.0, Europa, Galileo, Ganymede, and Helios. Figure 2 shows the time periods for the data collected for each release.

We extracted the same set of change metrics, including identifying bug-fixes, refactorings, and changeset size. Table 1 lists the metrics we used in our study. A detailed description of these metrics is given in [25]. For pre-Europa releases, i.e. releases 2.0, 2.1, and 3.0, as in [39], we mined the CVS log data by looking for four and five digit strings matching the bug IDs. For Europa and later, we matched six-digit strings to bug IDs. A manual review of data instances showed that no entries containing the word "bug" existed which were not caught by this pattern match. Extracting the metric Refactorings followed Moser's approach, namely tagging all log entries with the word "refactor" in them. The Age metric was calculated by reviewing all CVS log data from 2001 onward and noting the timestamp of the first occurence of each file name.

To determine changeset size, we used the CVSPS tool [18]. This tool identifies files which were committed together and presents them as a changeset. Slight modifications to the tool were required to ensure that the file names produced in the changesets included the path information to match the file names produced by our rlog processing script.

We wrote custom scripts to parse the CVS logs, converting the log entries into an SQL database. This data, along with changesets, bugs, and refactorings, were used to compute the metric values for each file. Finally, Weka-formatted files (ARFF) were produced. Figure 3 provides an overview of this process.

To ensure that the data resulting from the various input sources all contained matching filenames (the key by which the data were combined), and covered the same time periods, a few on-the-fly modifications were necessary. In cases where a file has been marked "dead", it is often moved to the Attic in CVS. This results in an alteration of the file path, which we adjusted by removing all instances of the pattern "/Attic/" from all file paths.

An artifact of using the CVS rlog tool with date filtering is that files which contain no changes during the filter period will be listed as having zero revisions, with no date, author, or other revision-specific information. This is true even if the file was previously marked "dead" on a branch. Thus, rather than examining only the date range required for each

Table 1: List of Change Metrics [25]

Metric name	Description
REVISIONS	Number of revisions made to a file
REFACTORINGS	Number of times a file has been refactored
BUGFIXES	Number of times a file was involved in bug-fixing (pre-release bugs)
AUTHORS	Number of distinct authors that made revisions to the file
LOC_ADDED	Sum over all revisions of the number of lines of code added to the file
MAX_LOC_ADDED	Maximum number of lines of code added for all revisions
AVE_LOC_ADDED	Average lines of code added per revision
LOC_DELETED	Sum over all revisions of the number of lines of code deleted from the file
MAX_LOC_DELETED	Maximum number of lines of code deleted for all revisions
AVE_LOC_DELETED	Average lines of code deleted per revision
CODECHURN	Sum of (added lines of code - deleted lines of code) over all revisions
MAX_CODECHURN	Maximum CODECHURN for all revisions
AVE_CODECHURN	Average CODECHURN per revision
MAX_CHANGESET	Maximum number of files committed together to the repository
AVE_CHANGESET	Average number of files committed together to the repository
AGE	Age of a file in weeks (counting backwards from a specific release to its first appearance in the code repository)
WEIGHTED_AGE	$\frac{\sum_{i=1}^{N} Age(i) \times LOC\_ADDED(i)}{\sum_{i=1}^{N} LOC\_ADDED(i)}$ where *Age(i)* is the number of weeks starting from the release date for revision i and *LOC_ADDED(i)* is the number of lines of code added at revision i

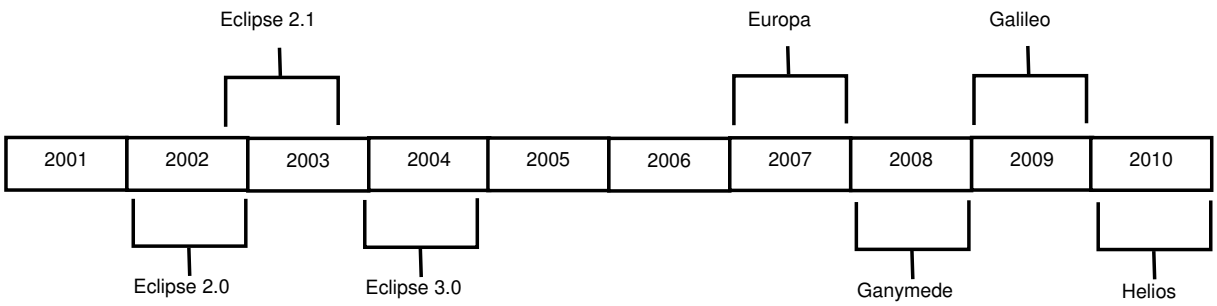

Figure 2: Data Timeline

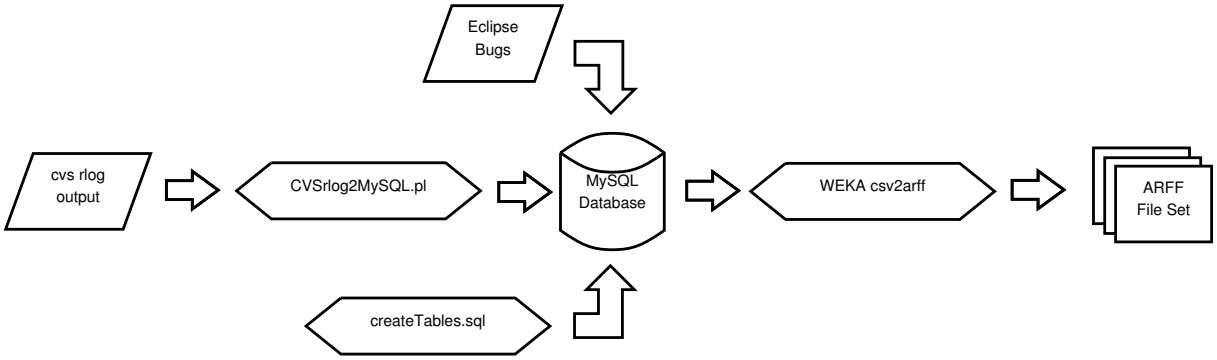

Figure 3: Data Collection Process

specific release, we obtained the rlog for the entire file history and determined the files which were alive and the revisions which applied to each release.

To validate our approach, we compared our resulting file set for the pre-Europa releases with the file sets available from Zimmermann's work [39]. We found that there were a few differences in the two data sets due to the independent data collection processes. While most of the files were common to both data sets, there was a small subset of files which were unique to each of them. For the three components, Platform, JDT and PDE, in the 2.0 release, we included 6893 files as compared to their 6730 files. In the 2.1 release, we had 7942 files while they had 7888, and in the 3.0 release, we had 10822 files as compared to 10593 in theirs. Further inspection showed that there were some differences in the list of plugins included in both studies. We also observed that some files which were not present in the earlier data set did have revisions during the development and production lifetime of the respective releases, and hence should have been included in the analysis. We thus included those in our data set.

3.2 Data Analysis

Using the features given in Table 1, we use the J48 decision tree learner to predict the files which are failure-prone. In this paper the prediction is done at release level, for each product in the product line, that is, both the training and testing data belong to the same release of the same product. Based on the confusion matrix shown below in Table 2, we define our performance measures along the lines of [25] and [39].

Table 2: Confusion matrix

		Predicted Class	
		Not Failure-prone	**Failure-prone**
True Class	**Not Failure-prone**	$n_{11}(TN)$	$n_{12}(FP)$
	Failure-prone	$n_{21}(FN)$	$n_{22}(TP)$

In the rest of this paper, we use the following measures of learner's performance.

- *PC (percentage of correctly classified instances, also known as accuracy)* $= (n_{11} + n_{22})/(n_{11} + n_{12} + n_{21} + n_{22}) * 100\%$
- *TPR (true positive rate, also known as recall)* $= n_{22}/(n_{21} + n_{22}) * 100\%$
- *FPR (false positive rate)* $= n_{12}/(n_{11} + n_{12}) * 100\%$

The measure PC, *Accuracy*, relates the number of correct classifications to the total number of files. The measure TPR, *Recall*, relates the number of files predicted and observed to be failure prone to the number of failure-prone files. It is also known as probability of detection. The measure FPR relates the files the learner incorrectly classified as failure-prone to the total number of non-failure-prone files. These measures are used by Moser, et al. [25].

The results were obtained using 10-fold cross validation. Apart from the prediction results obtained from the stratified cross-validation confusion matrix, we extract the tree produced by the J48 algorithm. The trees produced by the J48 algorithm were large in size with many leaves. For the Eclipse Classic product, the trees contained approximately 600 nodes and 300 leaves. It is manually difficult to parse the trees and determine the prominent predictors. For this purpose we modified the J48 code in Weka to output the gain ratio weights assigned to the nodes of the tree based on the number of correctly classified files from the total number of files. We then sorted the nodes in the descending order of weights and chose the nodes with the highest weight to be prominent predictors. All our data and results are available at [1].

4. OBSERVATIONS AS A SINGLE PRODUCT EVOLVES

In this section we discuss the performance of the machine learner and the sets of prominent predictors for single products in the product line.

4.1 How does our learner performance compare with previous results?

We performed classification of post-release failure-prone files for three older releases of Eclipse, namely 2.0, 2.1 and 3.0, similar to that performed by [25]. These older releases of Eclipse did not have many components. Platform, JDT and PDE were the important components, and the combination of these three components was distributed as Eclipse SDK. This combination of components is now one product called Eclipse Classic in the Eclipse product line. Moser, Pedrycz and Succi in [25] looked at three releases, 2.0, 2.1 and 3.0 of this single product. We performed classification on the same three releases for this product using the J48 learner.

Table 3 compares our results with theirs. We see that our results are slightly better than previous results across all three releases in terms of both correctly classified instances (PC) and false positive rates (FPR). Except for TPR values in two releases, 2.0 and 3.0, our results are similar to or slightly better than the previous results. On comparing the results for releases 2.0 and 2.1 with release 3.0, we see that both in [25] and in our study, 2.0 and 2.1 show better prediction results than 3.0, in terms of PC and FPR. This may be due to the fact that there was a restructuring of Eclipse's architecture in 3.0 which introduced more files and more changes.

Table 3: Comparison of classification performance for 2.0, 2.1, and 3.0 releases of Eclipse Classic

Release	Moser et al. [25]			This study		
	PC	TPR	FPR	PC	TPR	FPR
2.0	82	**69**	11	**88**	55	**5**
2.1	83	60	10	**85**	**63**	**9**
3.0	80	**65**	13	**84**	62	**9**

A reason for the difference in results may be the different number of files used, as described in Section 3.1. The data sets used in [25] consisted of significantly smaller subsets of the files in [39] (57% of the 2.0 files, 68% of the 2.1 files, and 81% of the 3.0 files), which was mentioned as due to incomplete history. Our data sets are larger, comparable in size to the data sets in [39], with a few differences between them.

4.2 Does learner performance improve as a single product evolves?

We next explore the performance for later releases of the same product, Eclipse Classic. The results for the years 2007, 2008, 2009 and 2010 in Table 4 show significant improvement in PC, TPR and FPR over the earlier years. One possible reason for the improved performance of the learner during the later releases could be the reuse of files implementing commonalities. For example, the 2009 Galileo release shows a true positive rate of almost 86% and a very low false positive rate of approximately 2%. This is important since a high true positive rate indicates that the files that have been predicted as failure-prone are indeed having failures. A high TPR can encourage projects to allocate resources before release or as early as possible after release in order to minimize failures. The very low false positive values indicate that there are few files predicted as failure-prone when they actually are not, which leads to saving resources that otherwise might be allocated unnecessarily to find defects which may not be present.

Table 4: Results for 2007-2010 releases of Eclipse Classic

Release	PC	TPR	FPR
3.3 (Europa)	93	79	4
3.4 (Ganymede)	94	81	3
3.5 (Galileo)	97	86	2
3.6 (Helios)	97	85	2

4.3 Is the set of predictors stable across releases of a single product?

We next explore whether the set of prominent predictors identified by the J48 algorithm remains stable across releases for a single product in the product line, namely Eclipse Classic. We refer to the top five predictors in the decision tree as the set of prominent predictors. Results reported by Zimmermann, Premraj and Zeller [39] for the three earlier releases showed that a model learned from one release could be applied to a later release without losing too much predictive power. However, their models had low TPR (i.e., recall), which made this potential reuse suggestive but not yet useful. Moser et al. [24] identified three change metrics that were among the top five defect predictors for releases 2.0, 2.1, and 3.0. These are *Max_Changeset*, *Bugfixes*, and *Revisions*.

For Eclipse Classic, Table 5 shows the prominent predictors from [25] and from our study. The predictors in bold are the ones that occur in all three of the early releases. Only *Revisions* is common to all three releases in both studies. In our study, *Weighted_Age* is common to all three releases, and *Age* and *CodeChurn* appear in two out of three releases. One interesting observation is that *Max_Changeset* which is one of the top three predictors in [25] does not appear even once as a prominent predictor in our study. This could be because of the difference in the two data sets.

The next step is to see if the same set of predictors appear for the later releases of the same product. It is not unusual in

Table 5: Comparison of Prominent Predictors

Release	Top five predictors from [25]	Top five predictors from this study
2.0	**Max_Changeset**, **Revisions**, Age, **Bugfixes**, Refactorings	**Revisions**, **Weighted_Age**, Ave_Changeset, Bugfixes, Max_LOC_Added
2.1	**Bugfixes**, **Max_Changeset**, **Revisions**, Max_LOC_Added, Max_LOC_Deleted	**Revisions**, CodeChurn, Age, **Weighted_Age**, LOC_Deleted
3.0	**Revisions**, **Max_Changeset**, **Bugfixes**, Age, Ave_LOC_Added	**Revisions**, Authors, **Weighted_Age**, CodeChurn, Age

machine learning for the best predictors to differ across data sets. Table 6 shows the five prominent predictors for the four later releases of the Eclipse Classic product. Notably, *Revisions* remains a common predictor in all four releases. Interestingly, *Max_Changeset*, which was not a prominent predictor in the early releases, appears as a common predictor in all four later releases. Also, *Weighted_Age*, which was a prominent predictor in the earlier three releases, appears in only two of the four later releases. Other predictors such as *Authors* appear in three releases. As the Eclipse Classic product evolves over time, *Revisions* and *Max_Changeset* continue to be the most prominent and stable predictors.

5. OBSERVATIONS AS THE PRODUCT LINE EVOLVES

In this section we discuss how the performance of the machine learner and the sets of prominent predictors change as the product line evolves.

5.1 Does learner performance improve as the product line evolves?

In addition to the Eclipse Classic product studied in section 4, we applied the learning algorithm to three other products in the Eclipse product line, Eclipse Java, Eclipse JavaEE, and Eclipse C/C++. Figures 4, 5 and 6 show the results for the PC, TPR and FPR across four years for the four products in Eclipse's product line. The X-axis shows the four products and the Y-axis shows the PC, TPR and FPR, respectively.

We observe that across the product line, results show an improving trend for all products. In terms of correctly classified instances, all products have PC rates above 90%, with the 2010 release of JavaEE having 98% correctly classified instances. The true positive rates lie in the range of 78% to 91%, with the 2009 Galileo release of the C/C++ product having the highest true positive rate of 91%. False positives show very low values, ranging from 4% to as low as 1%, with the 2010 Helios release of JavaEE product having the lowest FPR. It follows that every product shows improved prediction performance as the product line evolves.

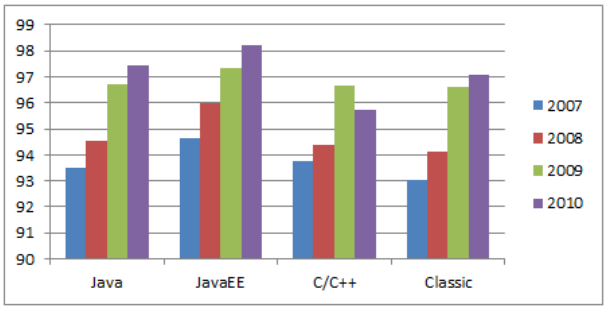

Figure 4: PC trends across products and releases

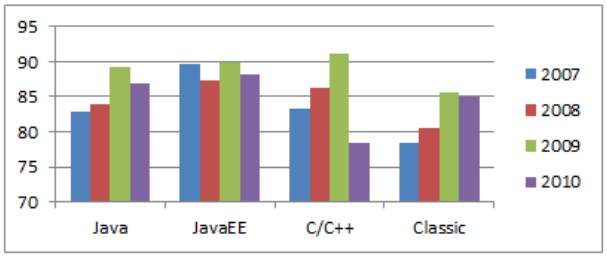

Figure 5: TPR trend across products and releases

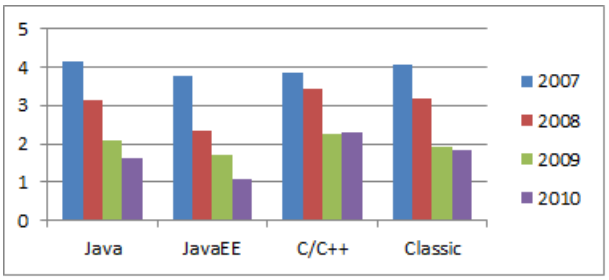

Figure 6: FPR trend across products and releases

5.2 Is the set of best predictors stable across products, as the product line evolves?

Earlier we saw that for Eclipse Classic, *Revisions* and *Max_Changeset* were stable, prominent predictors. Here we investigate if this set remains the same when we consider multiple products in the Eclipse product line or whether the set differs for different products in the product line.

Table 7 compares multiple products across the 2007-2010 releases. The last row of each column gives a summary of the prominent predictors for that particular product and in how many releases it appeared. We find that there is no common predictor across each product and each release. *Max_Changeset* appears as the most common predictor. *Revisions* and *Authors* are the other prominent predictors for the product line. Other predictors such as *CodeChurn*, *Bugfixes* and *Age* are prominent predictors for only one product.

The observations suggest that while there are some predictors which emerge as prominent ones across all releases of all products, others are prominent only for a single product or a few products. Hence we cannot conclude that there

Table 6: 2007-2010 Prominent Predictors for Eclipse Classic

Release	Top five predictors
3.3 (Europa)	Ave_Loc_Added, **Revisions**, Authors, Max_Changeset, Weighted_Age
3.4 (Ganymede)	**Revisions**, Age, Ave_Changeset, Max_Loc_Added, Max_Changeset
3.5 (Galileo)	Loc_Added, Authors, Max_Changeset, **Revisions**, Weighted_Age
3.6 (Helios)	Authors, **Revisions**, Max_Changeset, Bugfixes, CodeChurn

is a single set of predictors which is most prominent for all products.

6. DISCUSSION OF THE RESULTS

The highlights of the observations about prediction of post-release failure-prone files using change metrics in the Eclipse product line are summarized as follows:

1. Previous work [25] showed change metrics to be good predictors for predicting non-trivial post-release defects for older releases of Eclipse. We observe that change metrics continue to be good predictors for later releases of the same product. In fact, the results improve significantly as the product evolves with time. Predictions show high true positive and very low false positive values, better than previous studies.

2. All products show improvement in prediction of failure-prone files (i.e., files with bugs detected post-release) across releases.

3. The change metrics *Max_Changeset*, *Revisions*, and *Authors*, appear as prominent predictors across all four products in the product line.

4. There is a small set of change metrics that are prominent predictors for specific products in the product line. For example, *CodeChurn*, *Bugfixes* and *Age* appear to be prominent predictors of failure-proneness for Eclipse Java, Eclipse JavaEE and Eclipse C/C++ respectively.

Briefly, the results show significant improvement in the true positive and false positive rates across four products in the Eclipse product line over a period of seven years. During later stages of evolution, the capability to predict non-trivial post-release bugs shows improvement. As compared to the older releases (2.0, 2.1 and 3.0) the more recent releases show less activity in terms of number of *Revisions*. That is, a large portion of the total files in each product remain unchanged. This may allow the prediction algorithm to predict defects more accurately.

7. THREATS TO VALIDITY

This section discusses the internal and external validity of the study.

One threat to the internal validity of this study is that we performed analysis using only one machine learning algorithm, namely J48. Moser et al. [25] also used Naïve Bayes and logistic regression learners but found J48 to give the best results.

Another threat to the internal validity is the limited number of releases in the study. While analyzing more releases might give additional insight into the trends, the 2007-2010 releases provide a representative picture of the current product line situation. We did not include the minor quarterly releases into our analysis because there were fewer users downloading them and because the entries in the bug database for these minor releases were missing data for several components. Some of the minor releases reported higher numbers of failures while others did not report any. We plan to observe future releases as they become available and incorporate the new data for analysis.

As mentioned by [25], a possible threat to the internal validity could be the choice of metrics used in this study. We followed [25] in using a particular set of change metrics. In general, there could be other change metrics which may lead to different results. There may also be other yet undiscovered code metrics which may give better results.

There may also be inaccuracies in our data collection process at one or more steps. We performed manual and automated inspections on our data set to verify and validate its accuracy. We have made our data set public so that other researchers can validate the results of this study.

An external validity threat to this study is the extent to which these observations can be generalized to other product lines. Eclipse is a large product line with many developers in an open-source, geographically distributed effort. This means that the development of Eclipse product line is probably more varied in terms of the people involved and the development techniques used than in commercial product lines. Chastek, McGregor and Northrop consider the open-source development to be largely beneficial in terms of quality [6]. We hope to study other open-source software product lines and currently are studying a commercial software product line to learn more about reuse, change and reliability in product lines.

8. RELATED WORK

There have been few studies that consider defects in software product lines or mine their failure databases. As noted earlier, the lack of available datasets for product lines has seriously hampered investigation of the relationships between evolution and product quality, including the ability to predict failure-proneness. Of course, this is true not only for product lines. For example, Catal and Diri recently reported that only 31% of the 74 papers they reviewed used public datasets, making it difficult to reproduce or extend results [5].

With regard to product lines, Mohagheghi and Conradi [22], [23], compared the fault density and stability (change) of the reused and non-reused components in a system developed using a product family approach. They found that reused components have lower fault density and less modified code as compared to non-reused components.

Studies reported in [39], [26], [27], [38], [11] have used bug reports and bug repositories such as Bugzilla for predicting defects and failures. Jiang, Menzies, Cukic and others

Table 7: 2007-2010 Prominent Predictors for multiple products

Release	Java	JavaEE	C/C++	Classic
2007	Revisions, CodeChurn, Ave_Changeset, Weighted_Age, Max_Changeset	Authors, Max_CodeChurn, Max_Changeset, LOC_Added, Ave_Changeset	Authors, Revisions, Max_Changeset, Age, Max_CodeChurn	Ave_LOC_Added, Revisions, Authors, Max_Changeset, Weighted_Age
2008	CodeChurn, Age, Ave_LOC_Added, Authors, Max_Changeset	Authors, Revisions, Max_Changeset, Weighted_Age, Bugfixes	CodeChurn, Authors, Revisions, Ave_LOC_Added, Age	Revisions, Age, Ave_Changeset, Max_LOC_Added, Max_Changeset
2009	Authors, Revisions, Max_Changeset, Age, LOC_Deleted	Authors, Revisions, Max_Changeset, Bugfixes, Ave_Changeset	Authors, Revisions, Max_Changeset, Age, Ave_Changeset	LOC_Added, Authors, Max_Changeset, Revisions, Weighted_Age
2010	Authors, Revisions, Bugfixes, Max_Changeset, CodeChurn	Max_CodeChurn, Authors, Max_Changeset, Revisions, Bugfixes	Ave_CodeChurn, max_LOC_Added, Revisions, Ave_Changeset, Max_Changeset	Authors, Revisions, Max_Changeset, Bugfixes, CodeChurn
	Max_Changeset:4 **Revisions**:3 **Authors**:3 **CodeChurn**:3	**Authors**:4 **Max_Changeset**:4 **Revisions**:3 **Bugfixes**:3	**Revisions**:4 **Authors**:3 **Max_Changeset**:3 **Age**:3	**Revisions**:4 **Max_Changeset**:4 **Authors**:3

[14], [19] have used machine learning algorithms successfully to perform defect prediction. Ostrand, Weyuker and Bell were able with high accuracy to predict the number of faults in files in two large industrial systems [28]. Menzies et al. found that a lower number of training instances provided as much information as a higher number for predicting faulty code modules [20]. Zhang predicted the number of future component-level defects reasonably well using a polynomial regression-based model built from historical defect data [37].

Besides the work of [39] and [25] described previously, several different approaches for defect prediction also have used Eclipse as the case study, giving additional insights into the role of various code and process metrics in the Eclipse product line. D'Ambros, Lanza and Robbes analyzed three large Java software systems, including Eclipse JDT Core 3.3, using regression modeling, and found correlations between change coupling (files that change together) and defects [7]. They found that Eclipse classes have, on average, many more changes and more shared transactions than classes in the other two systems studied. Kim, Cai and Kim recently found that the number of bug fixes in three large open-source systems, one of them Eclipse JDT, increases after refactorings [15]. Schroter, Zimmerman and Zeller found that their predictive models (regression models and support vector machines) trained in one version can be used to predict failure-prone components in later versions (here, from version 2.0 to 2.1 of Eclipse) [33]. Eaddy et al. found a moderate to strong correlation between scattering (where the implementation of a cross-cutting concern is scattered across files) and defects for three case studies, one of which was an Eclipse component [8]. Shihab et al. reported work to minimize the number of metrics in their multivariate logistic regression model [34]. In a case study on the Eclipse dataset in [38], they identified four code and change metrics. One change metric, i.e., total prior changes in the 6 months before the release, was in their short list.

There has been a significant amount of work in the area of fault-proneness and/or failure-proneness prediction (often referred to as defect prediction) for both open-source and commercial software. With regard to open-source systems, Mockus, Fielding and Herbsleb [21] investigated the effectiveness of open-source software development methods on Apache in terms of defect density, developer participation and other factors. They showed that for some measures of defects and changes, open-source systems appear to perform better while for other measures, the commercial systems perform better. Paulson, Succi and Eberlein [29] compared the growth pattern of open-source systems with that for commercial systems. They found no significant difference between the two in terms of software growth, simplicity and modularity of code. They found, however, that in terms of defect fixes, open-source systems have more frequent fixes to defects. Rahmani, Azadmanesh and Najjar compared the prediction capability of three reliability models on failure reports for five open source software systems, finding that the failure patterns for open-source softwares follow a Weibull distribution [31].

With regard to commercial systems, Fenton and Ohlsson [9] related the distribution of faults to failures and the predictive accuracy of some widely used metrics. They found that pre-release faults are an order of magnitude greater than the operational failures in the first twelve months. Lutz and Mikulski [17] analyzed safety-critical anomalies in seven spacecraft and found that serious failures continue to occur with some frequency during extended operations. Recently, Hamill and Goševa-Popstojanova [13] conducted a study of two large systems to identify the distribution of different types of software faults and whether they are localized or distributed across the system. They analyzed different categories of faults and their contribution to the total number of faults in the system. Børretzen and Conradi [4] performed a study of four business-critical systems to investigate their fault profiles. They classified the faults into multiple categories and analyzed the distribution of different types of faults.

Finally, Nagappan, Ball and Zeller have shown that predictors obtained from one project are applicable only to similar projects [26]. Products in a product line are certainly

similar (i.e., share commonalities), but further investigation is needed to understand under what circumstances predictors from one product in a product line are relevant to other products in the product line.

9. CONCLUSIONS

The work reported here considers Eclipse as an evolving product line and distinguishes evolution of a single Eclipse product (Eclipse Classic, Eclipse Java, Eclipse Java EE and Eclipse C/C++) from evolution of the Eclipse product line. We study the performance of the J48 learner across these two evolution dimensions for a large set of change metrics available in Eclipse. The motivating questions are: (1) whether there are any change metrics that serve as good predictors for individual products as they mature over time, (2) whether any of these change metrics also serve as good predictors across all these products as the product line matures over time, and (3) whether performance improves, i.e., whether the ability to predict failure-prone files improves as the products mature over time.

The results are mixed, but generally positive. The answer to the first question is that there are change metrics that perform very well at predicting failure-prone files for the four Eclipse products studied over the seven releases studied, spanning 2002-2010. For the first Eclipse product (Eclipse Classic), the predictor performance was better than in a previous study for the early releases and improved across the later releases. Similarly for the other three products, the predictor performance improved for each product as it evolved across releases.

The answer to the second question is more complicated. For each product there was a small, stable set of change metrics that remained the prominent defect predictors as it evolved. For all products across all the releases, the change metrics "maximum changeset", "number of revisions" and "number of authors" were in the set of good predictors. Although the set of good predictors tended to be stable within each product as it evolved, there were some differences among the products regarding which change metrics were in the set of good predictors. Thus, while a small set of ten change metrics could be identified as prominent across the product line, only three change metrics were common to all products.

The answer to the third question is that the predictions of failure-prone files, both for single products and for the product line, tended to improve over time. For each of the four products studied, the predictor performance showed an overall improvement for accuracy, recall, and false positive rate. As the product line evolved, accuracy reached above 95% in 2010 for all products, while recall trended upward to between 78% and 87% for the products, and the false positive rate decreased nearly uniformly to 1-2%.

It is still unclear whether it will become possible to accurately predict failure-prone files across the products in an evolving product line. The high level of reuse in product lines which encourages that hope is offset by the on-going change and failures seen even in files implementing commonalities. The results of the current study suggest that further investigation of failure prediction in both open-source and proprietary product lines may yield a better understanding of how evolution of individual products affects the prediction of failure-prone files within product lines.

10. ACKNOWLEDGMENTS

This work was supported by National Science Foundation grants 0916275 and 0916284 with funds from the American Recovery and Reinvestment Act of 2009.

11. REFERENCES

[1] Promise 2011 data from this study. `http://www.cs.iastate.edu/~lss/PROMISE11Data.tar.gz`.

[2] Software Engineering Institute, Software Product Lines. `http://www.sei.cmu.edu/productlines/`.

[3] D. S. Batory, D. Benavides, and A. R. Cortés. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49(12):45–47, 2006.

[4] J. A. Børretzen and R. Conradi. Results and experiences from an empirical study of fault reports in industrial projects. In *PROFES 2006. LNCS*, pages 389–394. Springer, 2006.

[5] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

[6] G. Chastek, J. McGregor, and L. Northrop. Observations from viewing Eclipse as a product line. In *Proceedings on the Third International Workshop on Open Source Software and Product Lines*, pages 1–6, 2007.

[7] M. D'Ambros, M. Lanza, and R. Robbes. On the relationship between change coupling and software defects. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, WCRE '09, pages 135–144, Washington, DC, USA, 2009. IEEE Computer Society.

[8] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho. Do crosscutting concerns cause defects? *IEEE Trans. Software Eng.*, 34(4):497–515, 2008.

[9] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. on Software Engineering*, 26:797–814, 2000.

[10] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[11] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE'10, pages 495–504, New York, NY, USA, 2010. ACM.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.

[13] M. Hamill and K. Goševa-Popstojanova. Common trends in software fault and failure data. *IEEE Trans. Softw. Eng.*, 35:484–496, July 2009.

[14] Y. Jiang, B. Cukic, and T. Menzies. Can data transformation help in the detection of fault-prone modules? In *Proc. of the 2008 workshop on Defects in large software systems*, DEFECTS '08, pages 16–20, New York, NY, USA, 2008. ACM.

[15] M. Kim, D. Cai, and S. Kim. An empirical investigation into the role of API-level refactorings during software evolution. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*, 2011, To appear.

[16] S. Krishnan, R. Lutz, and K. Goševa-Popstojanova. Empirical evaluation of reliability improvement in an evolving software product line. In *Mining Software Repositories, MSR*, 2011, To appear.

[17] R. R. Lutz and I. C. Mikulski. Empirical analysis of safety-critical anomalies during operations. *IEEE Transactions on Software Engineering*, 30:172–180, 2004.

[18] D. Mansfield. Cvsps-patchsets for cvs. `http://www.cobite.com/cvsps/`.

[19] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engg.*, 17:375–407, December 2010.

[20] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, PROMISE '08, pages 47–54, New York, NY, USA, 2008. ACM.

[21] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the Apache server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 263–272. ACM Press, 2000.

[22] P. Mohagheghi and R. Conradi. An empirical investigation of software reuse benefits in a large telecom product. *ACM Transactions on Software Engineering and Methodology*, 17:13:1–13:31, June 2008.

[23] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 282–292, Washington, DC, USA, 2004. IEEE Computer Society.

[24] R. Moser, W. Pedrycz, and G. Succi. Analysis of the reliability of a subset of change metrics for defect prediction. In *ESEM*, pages 309–311, 2008.

[25] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *International Conference on Software Engineering*, pages 181–190, 2008.

[26] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 452–461, New York, NY, USA, 2006. ACM.

[27] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *ISSRE*, pages 309–318, 2010.

[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.*, 31:340–355, April 2005.

[29] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30:246–256, 2004.

[30] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[31] C. Rahmani, A. Azadmanesh, and L. Najjar. A comparative analysis of open source software reliability. *Journal of Software*, 5:1384–1394, December 2010.

[32] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[33] A. Schröter, T. Zimmermann, and A. Zeller. Predicting component failures at design time. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, ISESE '06, pages 18–27, New York, NY, USA, 2006. ACM.

[34] 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 G. Succi, M. Morisio, and N. Nagappan, editors, *ESEM*. ACM, 2010.

[35] Z. Stephenson. *Change Management in Families of Safety-Critical Embedded Systems*. PhD thesis, University of York, Mar. 2002.

[36] D. M. Weiss and C. T. R. Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[37] H. Zhang. An initial study of the growth of Eclipse defects. In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 141–144, New York, NY, USA, 2008. ACM.

[38] T. Zimmermann, N. Nagappan, and A. Zeller. *Predicting Bugs from History*, chapter Predicting Bugs from History, pages 69–88. Springer, February 2008.

[39] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, May 2007.