An Explanatory Analysis on Eclipse Beta-Release Bugs Through In-Process Metrics

Ayse Tosun Misirli
Department of Computer Engineering
Bogazici University
34342, Istanbul, Turkey
+90 212 359 7227
ayse.tosun@boun.edu.tr

Brendan Murphy[1], Thomas Zimmermann[2]
Microsoft Research
[1]Cambridge, UK, [2]Redmond, USA
bmurphy@microsoft.com, tzimmer@microsoft.com

Ayse Basar Bener
Ted Rogers School of Information Technology Management
Ryerson University
Toronto, CA
ayse.bener@ryerson.ca

ABSTRACT
Failures after the release of software products are expensive and time-consuming to fix. Each of these failures has different reasons pointing into different portions of code. We conduct a retrospective analysis on bugs reported after beta release of Eclipse versions. Our objective is to investigate what went wrong during the development process. We identify six in-process metrics that have explanatory effects on beta-release bugs. We conduct statistical analyses to check relationships between files and metrics. Our results show that files with beta-release bugs have different characteristics in terms of in-process metrics. Those bugs are specifically concentrated on Eclipse files with little activity: few edits by few committers. We suggest that in-process metrics should be investigated individually to identify beta-release bugs. Companies may benefit from such a retrospective analysis to understand characteristics of failures. Corrective actions can be taken earlier in the process to avoid similar failures in future releases.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics.* G.3 [**Mathematics of Computing**]: Probability and Statistics – *nonparametric statistics*

General Terms
Measurement, Reliability, Experimentation.

Keywords
In-process metrics, retrospective analysis, beta-release bugs.

1. INTRODUCTION
Software engineering is a complex discipline consisting of three aspects: product, process and resource (including people and tools). Each aspect should be carefully organized to develop reliable and high quality software products. Software testing is prioritized as the most critical phase that constitutes majority of the development costs [5]. This phase is supported by different VV activities as well as intelligent models [8, 10, 14] in order to detect as many faults (i.e., defects) as possible prior to the release. Nevertheless, the majority of software development projects, such as Eclipse, Mozilla, or Debian, are receiving bug reports from the customers after their releases.

Failures after the release of a software product are the most expensive and time consuming ones in terms of the effort and cost spent for fixing those [5]. Understanding characteristics of these failures would help software managers take corrective actions during the development and testing process and, hence, improve product quality. Previous studies identified various process [10], product [8] and organizational [14] metrics as the indicators of post-release failures. These studies focused on building predictive models to estimate defect-prone components in the system by using metrics and defect data from prior releases. A retrospective approach, on the other hand, would investigate and explain what went wrong during development and/ or testing periods of a software product leading to failures at the customer side. Practitioners may also benefit from such analysis to take corrective actions early on in the process and improve their development processes.

In this study, we investigate the development process of two Eclipse releases to understand unique trends/characteristics in software modules that have failures after the beta release. These failures are often reported by users who get the latest release of the software product. They are also collected from stack trace messages stored through exceptions that cause the system fail. Each failure in the system can point different portions of the software code. Different portions of the software can also have different characteristics in terms of development and testing processes, and these differences may explain these failures. Our primary research objective in this study is to find unique characteristics of files with beta-release bugs. Using a retrospective approach, we aim to explain the main characteristics of beta-release bugs in terms of development related process metrics.

In the next section, we describe previous studies on this topic. Then, we explain our methodology on data mining and metrics extraction (Section 3). After giving the details of our analysis in Section 4, we present our results with a discussion on each metric in Section 5. Finally, we define the threats to the validity of our results (Section 6) and conclude our paper.

Conference'10, Month 1–2, 2010, City, State, Country.

1.1 Contributions

We have made three main contributions with this study:

Categorization of beta-release bugs: We mined Eclipse bug database to identify bugs reported after the beta releases of Eclipse 2.1 and 3.0. We defined beta-release bugs as those reported with active stack traces, i.e. exceptions [4], after the beta release in Eclipse. Our analysis on bug database shows that very few (3-5%) files are associated with bugs reported after beta-release, and hence, it is hard to distinguish them from other files.

Identification of process metrics: After categorizing files according to their associated bug types, we observed Eclipse version database, which keeps all activities done on the main branch to define software factors related to the activities inside the development process. As a result, we identified six in-process metrics: Age, number of edits, number of committers, average changed lines of code, last edit date and average time between edits. Statistical analyses have been conducted for each attribute independently.

Characterization of files with beta-release bugs: Results of our experiments on different file categories corroborated the motivation of this study: Files with beta-release bugs have different characteristics in terms of in-process metrics. Our proposed analysis would help practitioners to interpret unique characteristics of these failures via in-process metrics. Therefore, they would take corrective actions to avoid similar failures in the upcoming releases.

2. RELATED WORK

There are numerous studies that investigate faults and fault distribution on software projects or build predictive models.

2.1 Understanding Software Faults

Studies analyzing the faults and software metrics characterizing these faults are mostly conducted on different commercial software systems. Moller and Paulish presented an empirical investigation of software fault distribution [2]. They observed that certain modules in a commercial software product have greater concentration of faults. Results conclude that modules with faults are highly correlated with new and changed LOC, module size and programming language.

Later, Fenton and Ohlsson conducted a quantitative analysis on faults and failures in two major releases of a large legacy system [3]. Based on their analysis, faults found in pre-release and operation phases are concentrated on a very small number of modules, but neither of these faults can be explained with size and complexity. Ostrand and Weyuker also followed the previous work by testing similar hypothesis on an inventory tracking system [7]. They found that severity of faults and the stage they are found have strong correlations with Pareto principle. In contrast to previous study [3], they found that small and younger files have higher fault densities.

In summary, these studies are retrospective in the sense that authors aim to understand distributions of fault types by product and process metrics on closed software systems. They are similar to our research in terms of the approach used for analyses, while metrics were often limited to the product aspect (i.e., size, complexity) of a software.

2.2 Predicting Software Faults

Recently, defect prediction has become very popular in the field of software engineering. Different approaches, in terms of the algorithm, metrics and type of defects, are used to predict defect-proneness of software systems. Size and complexity metrics extracted from the source code are the most widely used measures [8]. Menzies et al. [8] reported that static code attributes are significant indicators for pre-release defects. Zimmermann et al. [11] extracted and publicized (in Promise repository [15]) code metrics from Eclipse releases 2.0, 2.1 and 3.0 to predict pre- and post-release defects.

Nagappan et al. [10] extracted process (changed lines of code, number of changes, and developers) and product (cyclomatic complexity, code coverage) metrics from Microsoft Server 2003 and XP for fault prediction. Furthermore, the metrics set is enriched with code, churn, dependencies and organizational metrics for Windows Vista [14]. Results indicate that organizational metrics (number of engineers, ex-engineers, organizational code ownership) are the most significant indicators of post-release failures.

These studies enriched the set of software metrics by adding process-related data and improved predictive models. They are a form of prospective studies in which it is required to *make estimates of an outcome* with fewer potential sources of bias and confounding effect, compared to a retrospective study [9]. Despite more proneness to experimental biases, a retrospective analysis is quite powerful with statistical tests in *interpretation of the predictive value* of software metrics and *their explanatory effects* for faults. Such an analysis requires more complex and rich data to make accurate explanations via statistics. Thus, we have used an open source project, Eclipse, which is one of the largest and richest in terms of development data.

3. DATASET

We examined two Eclipse releases, namely 2.1 and 3.0, and their development process. In this section, we explain the formal release schedule and bug categorization to match bugs with their corresponding release and process metrics used in our study.

3.1 Eclipse Release Schedule

Eclipse has very formal release plans such that, every year in June, its software team announced a major Eclipse release. During the development period of each release, they have stable builds in every six weeks. Stable builds (milestones) are integration builds which are found stable after a few days trial by the architecture and test teams. The latest milestone represents the "beta release" in a typical software release lifecycle, such that users who would like to get latest features and bug fixes are welcome to download the latest stable build and report any problems until major release.

Previously, Schröter et al. mined Eclipse bug and version databases to match revisions stored in CVS with bugs in package and class level [13]. Later, Zimmermann et al. published pre- and post-release defects for files in the Eclipse releases 2.0, 2.1 and 3.0 [11]. Our goal is to observe revisions made for files with beta-release bugs in releases 2.1 and 3.0 *during their development periods* in order to investigate their reasons. We know which files are being developed and released in 2.1 and 3.0 based on the data in Promise repository [15]. Therefore, we first defined the start dates of the development periods for releases 2.1 and 3.0 as well

as their milestones by looking at Eclipse project plans [12]. We could not use Eclipse 2.0 data for our analysis, because its start date and milestone periods were not available in [12]. Release 2.1 has a start date on August 30th, 2002, and release on March 27th, 2003. In between, it has 4 milestones until the beta release (February 7th, 2003). Similarly, 3.0 has a start date on March 28th, 2003 and beta release on May 21st, 2004 before its release on June 27th, 2004. Beta-release bugs for each release are started to be reported after the latest milestone, prior to which, the revisions and bug fixes are done. We observed the revisions for all files in a release from its start date until beta-release to make an explanatory analysis on bugs reported after beta release.

3.2 Categorization of Beta-Release Bugs

Beta-release bugs are different in terms of the content of their bug reports and reporters. In this study, we categorized beta-release bugs based on reports coming from active stack traces, i.e., exceptions. These exception reports are thrown by the system when an unexpected situation occurs during when users/ developers are actively using the beta version of Eclipse. We used the structural information extracted from Eclipse bug reports in the work of Bettenburg et al. [4] in order to find "bugs with stack traces". We classified files for releases 2.1 and 3.0 based on their associated *bug reports*:

1. Files with *bugs* reported *after beta-release* in the form of *active stack traces*
2. Files with *bugs* reported *prior to beta-release*
3. Files with *no bugs*

Table 1 summarizes statistics for categorization of Eclipse files (in disjoint sets) in releases 2.1 and 3.0. Based on this categorization, we formed three file sets and conducted our analysis on all file sets to understand how process metrics specifically change for files with beta-release bugs.

Table 1. File categorization in terms of bug reports

# Files (%)	**2.1**	**3.0**
with *beta-release bugs* including *stack traces*	173 (2%)	193 (2%)
with bugs reported *prior to beta-release*	1729 (22%)	2611 (25%)
with *no* bugs	5947 (76%)	7727 (73%)

3.3 Metric Extraction and Hypotheses

We selected six in-process metrics, four of which were directly used in previous studies investigating their relationships with fault proneness [7, 10, 13, 14], *age, number of edits, number of committers, average changed lines of code*, and the rest of them, i.e. *last edit date, average time between edits*, are recently defined based on a previous study on changes during the development phase and their time dependencies [6]. For each release (2.1 and 3.0) in Eclipse, we formed three datasets containing in-process metrics for three file sets (1,2,3 above).

During our experiments, the first set represents *files with beta-release bugs*, whereas the second represents *files with other bugs*. Since the sizes of three datasets are different, we transformed these datasets into frequencies, i.e., 200 bins were formed, where each bin contained the number of files taking a value within a specified range, to allow a comparable analysis between the distributions of file sets. As the second step, we removed files with no activity, in terms of commits and bug assignments, during the development periods of 2.1 and 3.0, because they distorted the distribution of data due to high peaks at the value of 0. This procedure did not introduce a bias into our study due to inactivity of these files. Furthermore, we were able to identify distributions and peaks more easily.

In the third step, we observed the distributions of in-process metrics and we had seen that some of the distributions had very long tails due to outliers in data. To avoid long tails in the distributions, we re-defined the number of bins for all file sets by defining a cut-off value that removed top 5% of data in files with user bugs. This approach helped us to interpret the distributions better. Finally, we formed hypotheses for six in-process metrics to test their explanatory effect on different file sets using statistical analyses.

Age: We selected files created before beta release and calculated the age in terms of days since their creation date until the beta release. Our aim was to observe whether young files are more fault-prone than old files, as in the work of Ostrand and Weyuker [7]. Thus, we formed the following hypothesis:

H1: As the age of a file increases, then it is likely to have fewer bugs after beta release assigned to the file.

Number of Edits: We counted the number of edits done on files during the development periods of 2.1 and 3.0 prior to their beta releases. For instance, we observed the development activity between August 30th, 2002 and the final beta release date, February 7th, 2003 to count the number of edits on files in release 2.1. This metric was previously used in other defect prediction studies [10, 13] and found to be significant indicators of software defects. Therefore, our hypothesis was:

H2: As the number of edits increases on a file, it is expected that the file will have more bugs after beta release.

Last Edit Date: This metric count the days between the dates when a file is last checked into CVS and the beta release. We believe that if a file is edited recently, then there is a possibility that it may not be tested enough before the beta release. We computed the time (in days) since the latest edition for all files in releases 2.1 and 3.0. Our hypothesis was that:

H3:As a file is edited recently (closer to release date), then it is more likely to have bugs after beta release.

Number of Committers: A previous study done on binaries of Windows Vista showed that organizational metrics such as the number of developers (i.e., committers) are statistically significant indicators of post-release defects with the best prediction accuracy, compared to churn and code metrics [14]. Therefore, we counted the number of unique committers who edited files during the development periods of 2.1 and 3.0. As a similar trend with the number of edits, we formed our hypothesis as follows:

H4: The more developers make changes on a file, the more bugs after beta release will be reported on this file.

Average Changed Lines of Code: The number of edits is an important indicator of development activity, and the extent of these edits provide valuable information about how much change, i.e., added/deleted/modified lines of code, has been done on files.

We used this metric to measure the average amount of changes on files. The complexity of these changes would provide more information, but it could not be measured through CVS (without source codes). Therefore, we computed the changed LOC for all files during the development periods of 2.1 and 3.0 until the beta releases. Our hypothesis was:

H5: The more lines of code (LOC) is edited on a file, the more bugs after the beta release will be reported on this file.

Average Time between Edits: This metric aims to measure the independence between consecutive edits on the main branch, or conversely, dependencies between changes. If the time between several edits on a file is less than 2 days, then it is more likely that these edits may be interdependent due to a bug fix or feature development. As the time between edits on a file increases, there may be several independent tasks (developments due to different projects, requests, functionalities) completed on that file. Thus, we formed our hypothesis as follows:

H6: As the time between edits on a file increases (seldom revisions), the probability of this file having bugs after the beta release will increase.

4. STATISTICAL ANALYSES

Our main research objective is to find unique trends in files with beta-release bugs in terms of in-process metrics. To test our hypotheses we conducted three analyses each of which serves for a different purpose in this study:

Non-parametric Correlations between Files: We applied Spearman's rank correlation statistics on pairs of file sets (for each metric) in order to identify whether there is a interdependence between files with beta-release bugs, files with other bugs and files with no bugs. Spearman's rank correlation coefficient indicates a high correlation (close to 1) between two file sets, if the distributions of these files with respect to a software metric are not completely independent from each other. High correlation between files with beta-release bugs and other files indicates a statistically significant lack of independence between them, which would make it more difficult to distinguish files with beta-release bugs from other files. We cannot absolutely conclude these facts using only one significant test, because Spearman test is also insensitive to some kind of independences.

Mann-Whitney U-Test: This test checks whether two populations tend to be distributed differently with respect to a factor such that their medians are different from each other (one is shifted left or right) [1]. It is more sensitive to dependencies between two populations compared to Spearman's test. If Mann-Whitney test rejects its null hypothesis: "*two file sets come from distributions with equal medians*" with 95% confidence, this indicates that the distribution of files with beta-release bugs and the other file set with respect to a software metric (e.g. age) tend to concentrate on different medians although their distributions are highly correlated (according to Spearman's rank coefficient). These differences in terms of medians may help us distinguish files with beta-release bugs with respect to a specific metric.

Non-parametric Correlations between Attributes: In contrast to the analyses on metrics individually, pairs of metrics may be more explanatory to analyze files with beta-release bugs. In this test, we applied Spearman test on all possible attribute pairs to understand the dependencies between attributes. If the rank coefficient between two software metrics is higher than 0.70, this indicates that this metric pair should be observed to get a broader understanding on files with beta-release bugs. Thus, in the last analysis, we plotted scatter diagrams between metric pairs, who are strongly correlated, with respect to files with beta-release bugs, files with other bugs and files with no bugs, respectively.

5. RESULTS

Interpretations of the statistical tests are summarized for six in-process metrics. We named files with beta-release bugs reported in stack traces as *BS*, files with bugs reported prior to beta-release as *BNS*, and files with no bugs as *NB.*

5.1 Age

Spearman's rank correlation test for BS, BNS and NB are computed for Eclipse releases 2.1 and 3.0 (Table 2). The bold and gray-shaded cells in Table 2 indicate that selected pairs (BNS and NB) are significantly correlated in terms of *age* metric. We can also see that BS follow different trend from others (BNS and NB), i.e., mutual independence. However, BNS and NB are strongly correlated, i.e., it is very likely that there is an interdependence (positive or negative relation) between two file sets. We also plotted their distributions in Figure 1. Figure 1 shows that 14% of files with no bugs (NB) are around 4 months old in release 2.1. However, files with beta-release bugs (BS) tend to concentrate around 300 days, which means they are older than files with no bugs (NB). In release 3.0, it is hard to see whether all file sets share the same distribution, since they are concentrated on different time periods. However, the highest percentage of files with beta-release bugs (12%) is around 2 years old. This shows that our first hypothesis (H1) may not hold: *As the file gets older, it more likely contains bugs after beta release.*

To make a stronger claim, we applied Mann-Whitney U-tests between BS, BNS and NB. Results indicate files with beta-release bugs (BS) have different medians with 95% confidence (p-value < 0.05). That is; we can distinguish files with beta-release bugs (BS) from other files using age metric by looking at different medians they are concentrated on.

Table 2. Spearman Correlations: "Age"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	0.54	0.49		**BS**	1.00	0.56	0.55
BNS		1.00	**0.84**		**BNS**		1.00	**0.88**

Table 3. Spearman Correlations: "Number of Edits"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	**0.89**	**0.73**		**BS**	1.00	**0.73**	0.66
BNS		1.00	**0.86**		**BNS**		1.00	**0.90**

Table 4. Spearman correlations: "last edit date"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	**0.72**	0.64		**BS**	1.00	**0.71**	0.64
BNS		1.00	**0.91**		**BNS**		1.00	**0.92**

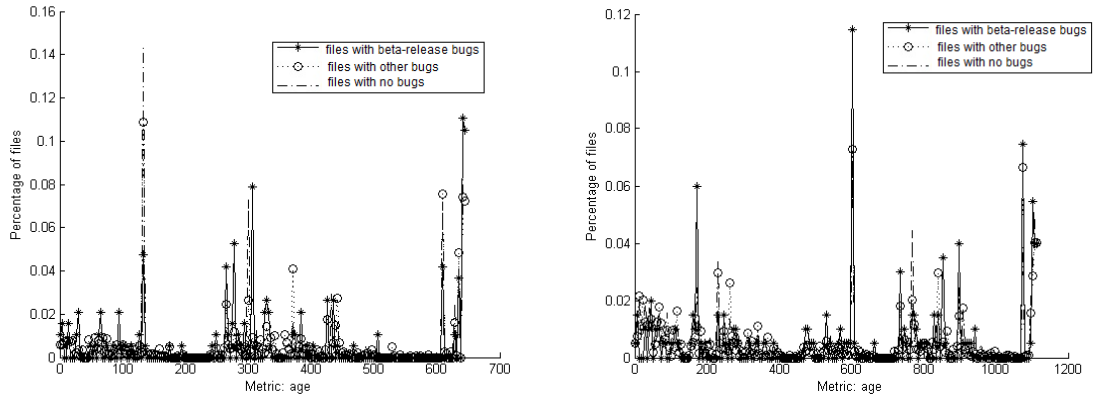

Figure 1. "Age" distributions in files with beta-release bugs (*-), files with other bugs (o-), and files with no bugs (.-): (a) release 2.1 (b) release 3.0. X axis shows the age in days, whereas Y axis shows the percentage of files.

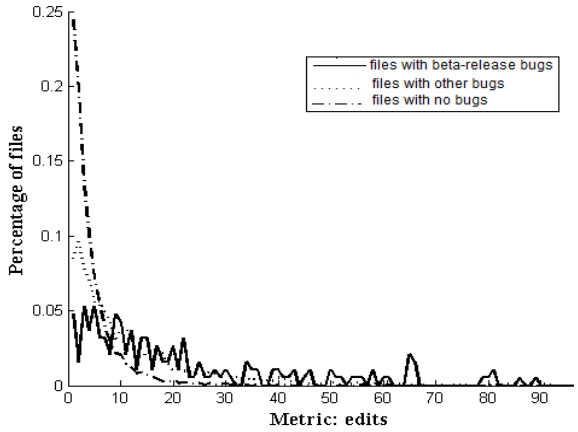

Figure 2. Distributions for "Number of edits"

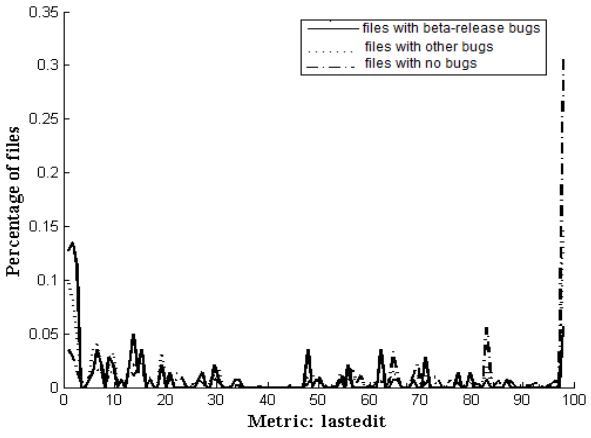

Figure 3. Distributions for "Last edit date"

5.2 Number of Edits

We applied the same procedure that we used for the *age* metric between BS, BNS, NB. Table 3 summarizes the Spearman correlation coefficients, where gray-shaded and bold cells indicate that there is a strong correlation between a pair with 95% significance. From Table 3, it is seen that files with beta-release bugs, have very strong relations with other files. This indicates that they follow a very similar trend and it is hard to distinguish BS in terms of distributions.

We also plotted the distribution of *number of edits* for BS, BNS, NB only for release 3.0 due to space limitations. From Figure 2, it is clear that files with other bugs (BNS) and no bugs (NB) follow smooth exponential distributions, where most of the files has less than 10 edits. However, distribution of files with beta-release bugs (BS) has heavier tails: Files in BS are edited more than other files.

We validated this finding with Mann Whitney U-tests for pairs of BS-BNS, BS-NB. The test rejects the null hypothesis "*files with beta-release bugs and the other file set come from distributions with equal medians*" with 99% confidence (p-value < 0.01). Therefore, files with beta-release bugs (BS) have similar distributions with other files (all follow exponential trends), but they have median shifts indicating that there are relatively more edits on them. As a result, our second hypothesis (*H2*) holds: *The more edits are done on a file, it is more likely that the file will contain bugs after beta release.* So *number of edits* is also a significant indicator for beta-release bugs in Eclipse releases 2.1 and 3.0.

5.3 Last Edit Date

For this metric, Spearman correlation coefficients (ρ) and distribution plots for BS against the other files (BNS, NB respectively) are shown. Correlations (gray-shaded and bold cells) in Table 4 show that files with beta-release bugs are strongly correlated with files with other bugs in terms of last edit date ($0.7 < \rho < 0.9$). We also plotted the trends for the files with beta-release bugs (BS) and the other file sets (BNS, NB) on release 2.1. Figure 3 shows that around 35% of NB are edited 95 days ago (nearly 3 months), whereas 11% of BS, i.e., the highest proportion of files, are edited a day ago (very recently compared to BNS, NB).

To support the existence of median shifts and distinguish BS from others, Mann Whitney U-tests were conducted. The test rejects the null hypothesis for BS-BNS, BS-NB pairs with 98% confidence (p-value < 0.02) indicating that files with beta-release bugs and other files may come from similar distributions, but they have

different medians (BS are edited more recently). Therefore, our third hypothesis (H3) also holds: *Files with beta-release bugs are edited closer to the release date.* So last edit date is also a significant indicator for beta-release bugs in Eclipse.

5.4 Number of Committers

To observe whether the *number of committers* working on files has a unique characteristic for files with beta-release bugs, we plotted the trends and computed Spearman correlation coefficients. Based on Table 5, gray-shaded and bold cells indicate strong correlations between BS-BNS and BS-NB for release 2.1. This indicates that files with beta-release bugs may come from similar distributions with other files or they are interdependent to each other. To explain this strong correlation, we plot distributions of BS, BNS and NB for releases 2.1 and 3.0. Since both releases show very similar trends, we present release 3.0 in Figure 4.

Figure 4 presents a typical scenario valid for various software projects such that the greatest portion of files are edited by only 1 or 2 committers. For instance, in release 3.0 (Figure 4), 65% of files with no bugs are edited by only one committer, whereas this percentage decreases in files with beta-release and other bugs. This argument, in fact, supports our forth hypothesis (H4): *As the number of committers working on files increases, then it is very likely that files will contain bugs after the beta release.* We can clearly see the median shifts for files with beta-release bugs. To support our claim, Mann-Whitney U-tests between BS and other file sets also rejected the null hypothesis with 95% confidence, meaning that BS has different medians from other file sets. Thus, number of committers is also a significant indicator for beta-release bugs.

5.5 Average Changed LOC

We again computed Spearman correlation coefficients between three file sets for this metric. Table 6 shows that all files are strongly correlated with each other in terms of their distributions. The distributions for release 3.0 in Figure 5 validate this fact (all files follow almost exponential distributions). However, Figure 5 also shows that more than 20 LOC were changed in 30% of files with beta-release bugs, in contrast to 45% of files with no bugs, which had less than 20 LOC being changed. Therefore, as in the case of *number of edits*, files with beta-release bugs have heavier tails (more changes in terms of LOC) than other file sets. We also conducted Mann-Whitney U-tests and found that distributions of files with beta-release bugs have shifted medians with 95% confidence . Based on Figure 5 and significance tests, we can support our hypothesis (*H5): Files with beta-release bugs are edited (in terms of LOC) more than other files.* Therefore, *average changed LOC* is also a significant indicator for files with beta-release bugs.

5.6 Average Time between Edits

The final metric is *average time between edits*. We applied the same procedures and found the correlations depicted in Table 7. From the table, it can be seen that all file sets except BNS-NB in release 3.0 have mutually independent from each other. They should follow very different distributions with no relations between one another. We plotted the trends to understand the actual reasons for these low correlations. Figure 6 shows the distributions for release 3.0 in terms of *average time between edits* metric. As seen in the figure, files with beta-release bugs follow a completely different trend from other file sets, which are somewhat similar to an exponential trend. Around 6% of files with beta-release bugs (as the highest concentration) are edited, on the average, every 260 days, i.e., 8.5 months. This is the opposite for files with other and no bugs, such that more than 20% files with no bugs are edited on the average every 46 days. Therefore, we can support our sixth hypothesis (*H6): If files are seldom edited (in every 8 months), then it is more likely that developers will introduce beta-release bugs into the code.* If they are frequently edited (every month), developers may produce more reliable code.

We also conducted Mann-Whitney U-tests and found that files with beta-release bugs (BS) have different medians from other files with 95% significance. Therefore, *average time between edits* is also a significant indicator of beta-release bugs. As a result, by investigating the attributes individually,

- We can conclude that all six metrics have explanatory effects on beta-release bugs: bugs reported after beta-release with stack traces.
- Although in most cases, all files are likely to share the same distribution in terms of in-process metrics, files with beta-release bugs have median shifts, which make them easy to distinguish from other files.

Table 5. Spearman Correlations: "Number of committers"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	**0.96**	**0.96**		**BS**	1.00	**0.40**	**0.27**
BNS		1.00	**0.94**		**BNS**		1.00	**0.96**

Table 6. Spearman Correlations: "Average changed LOC"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	**0.87**	**0.90**		**BS**	1.00	**0.89**	**0.90**
BNS		1.00	**0.99**		**BNS**		1.00	**0.98**

Table 7. Spearman Correlations: "Time between edits"

Release 2.1					Release 3.0			
	BS	**BNS**	**NB**			**BS**	**BNS**	**NB**
BS	1.00	0.14	0.04		**BS**	1.00	0.00	-0.02
BNS		1.00	0.48		**BNS**		1.00	**0.63**

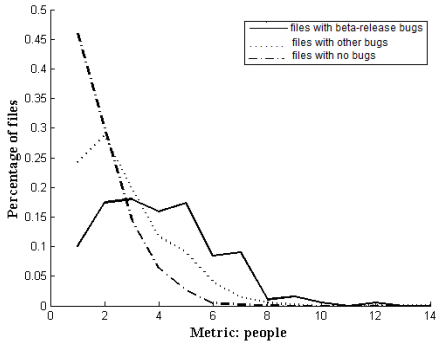

Figure 4. Distributions for "Number of committers"

Table 8. Spearman correlation coefficients (rho) between process metrics for files with bugs including stack traces.

Release 2.1						
	age	edits	lastedit	people	chgLOC	timeedits
age	1.00	0.08	0.11	0.12	0.13	0.19
edits		1.00	-0.14	**0.78**	0.69	**0.78**
lastedit			1.00	0.08	-0.01	-0.09
people				1.00	0.52	0.70
chgLOC					1.00	0.46

Release 3.0						
	age	edits	lastedit	people	chgLOC	timeedits
age	1.00	0.40	0.00	0.40	0.09	0.67
edits		1.00	-0.38	**0.78**	0.43	0.62
lastedit			1.00	-0.26	-0.16	-0.12
people				1.00	0.31	0.57
chgLOC					1.00	0.23

Edits: number of edits. People: number of committers. ChgLOC: average changed LOC. Timeedits: average time between edits. Bold cells indicate strong correlations ($\rho > 0.75$) with p-value less than 0.05.

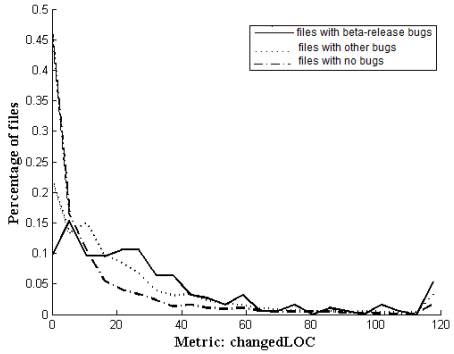

Figure 5. Distributions for "Average changed LOC"

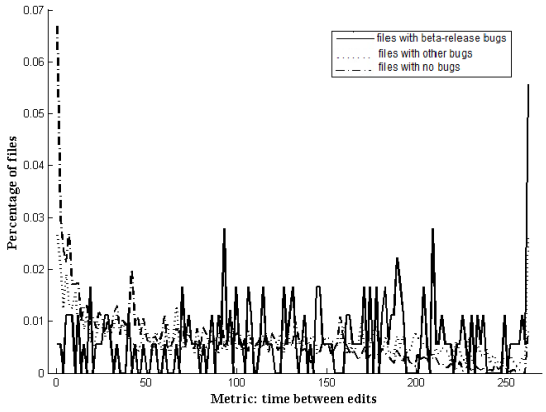

Figure 6. Distributions for "Average time between edits"

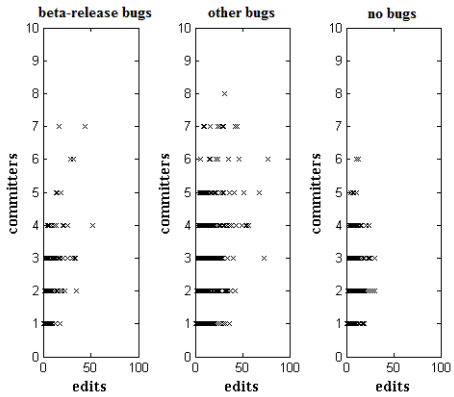

Figure 7. Scatter diagram for edits-committers in 2.1.

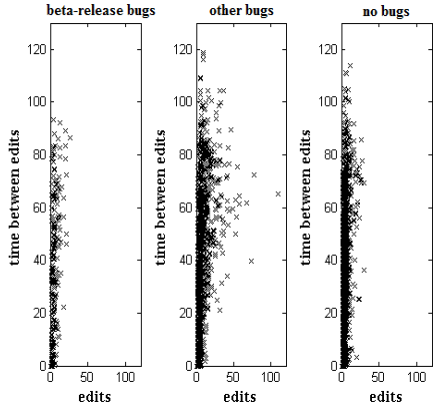

Figure 8. Scatter diagram for edits-timeedits in 2.1.

5.7 Correlation Between Process Metrics

We found that the six in-process metrics can be used as significant indicators of beta-release bugs independently. However, some of them may actually be related to each other, e.g.as the number of people editing files increases, then the number of edits would also increase. To observe these relations among in-process metrics with respect to files with beta-release bugs, files with other bugs and files with no bugs, we computed Spearman correlation coefficients on all metric pairs.

Table 8 summarizes the results on BS for releases 2.1 and 3.0. Gray-shaded and bold cells indicate significant strong correlations ($\rho > 0.75$) between metrics, such as number of edits (edits) and number of committers (people) (for releases 2.1 and 3.0) and, number of edits (edits) and average time between edits (timeedits) (for release 2.1).

To distinguish files with beta-release bugs from others using these metric pairs, we observed scatter diagrams (Figure 7 for *edits-people* and Figure 8 for *edits-timeedits)* for release 2.1. The plots are identically the same for release 3.0 except there are more activities in release 3.0. From Figure 7, we can see that although medians of three file sets are different from each other in terms of edits and people, files with beta-release bugs are concentrated on a smaller region, with less edits and fewer people, which is a subset of the region covered by files with other bugs, i.e., more edits, more people. Furthermore, files with no bugs have also less edits by few people. Therefore, if we built a rule-based system using the most correlated metric pairs, then we would identify only the files with other bugs by putting a lower threshold as **(30,**

5) for *(edits, people)*. Based on Figure 7, files edited more than 30 times and by more than 5 people are more likely to contain other bugs. However, files edited less than 30 times and by less than 5 people may either contain beta-release bugs or contain no bugs.

Figure 8 also shows a similar pattern: Files with beta-release bugs have less than 30 edits and edits within every 100 days. This pattern suggests that, in Eclipse, majority of beta-release bugs are mapped with files with little activity, rather than files with more activities; possibly the testing efforts in Eclipse focus primarily on files with more activity, which could explain why the beta-release bugs have been missed.

To summarize, although we were able to identify that in-process metrics follow unique trends for files with beta-release bugs, it is less likely to detect certain type of bugs with a single prediction model due to large overlaps in small regions of distributions. Thus, practitioners should use such simple and powerful statistical analyses to explain characteristics of certain defect categories separately. Using this information, the development process can be improved and certain actions can proactively be taken to avoid post-release failures.

6. THREATS TO VALIDITY

We overcame threats due to bug categorization by filtering bugs based on information reported in stack traces rather than using start and end dates of beta release. We also made assumptions during metric extraction and data filtering, such as removing long tails from the distributions, and removing files with no activity. It is unavoidable to eliminate all kind of biases in retrospective studies, since the goal is to mine a very rich and complex data, and apply statistics to represent the situation as good as possible. We removed files which are outliers and cause the long tails in distribution, since they did not represent the majority of the population. Finally, we put high attention to selection of significance tests, each overcomes threats that may occur due to the previous ones, e.g. Spearman test is insensitive to certain dependencies which can be handled by Mann Whitney U-test. Drawing general conclusions from retrospective analyses, is a real challenge, since every project depends on several variables. However, our analysis was conducted on one of the largest open source projects, and therefore, it can be easily replicated, refuted or improved.

7. CONCLUSION

Our analyses shows that:

1. When files with beta-release bugs can be observed independently from other files, it is seen that they have different characteristics in terms of process-related factors.
2. All files show a similar trend (in terms of their distributions) such that most of the files are edited less than 20 times and by less than 3 committers.
3. However, files with beta-release bugs are different in terms of how their population is shifted to less activity. They are edited not very frequently and less than 20 times, compared to files with other bugs (more than 20 edits by more than 3 committers).
4. Prediction models detecting all types of post-release defects may not successfully predict certain type of defects due to their unique in-process characteristics. Thus, we should concentrate on such defects to understand unique characteristics of these failures associated with these defects.
5. Retrospective analysis can help practitioners understand developments processes and change certain principles to avoid similar failures in future releases.

8. ACKNOWLEDGMENTS

Many thanks to Nicolas Bettenburg for sharing his data on stack traces in Eclipse bug reports.

9. REFERENCES

[1] Wilcoxon, F. 1945. Individual comparisons by ranking methods. Biometrics Bulletin, (1): 80-83.

[2] Moller, K.H., Paulish, D. 1993. An empirical investigation of software fault distribution, In Proceedings of the first software metrics symposium, IEEE CS Press, 82-90.

[3] Fenton, N., Ohlsson, N. 1999. Quantitative Analysis of Faults and Failures in a Complex Software System, IEEE Transactions on Software Engineering, (26): 797-814.

[4] Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S. 2008. Extracting structural information from bug reports, In Proceedings of the Fifth International working conference on Mining Software Repositories, 27-30.

[5] Brooks, A. 1995. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley.

[6] Alam, O., Adams, B., Hassan A. 2009. A Study of the Time Dependence of Code Changes, In Proceedings of the 16th working conference on Reverse Engineering.

[7] Ostrand T., Weyuker, E. 2002. The distribution of faults in a large industrial software system, In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, 55-64.

[8] Menzies T., Greenwald, J., Frank, A. 2006. Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering, (33): 2-13.

[9] StatsDirect Ltd. 2009. Statistical Help Basics, http://www.statsdirect.com/help/statsdirect.htm.

[10] Nagappan, N., Ball, T., Murphy, B. 2006. Using historical in-process and product metrics for early estimation of software failures, In Proceedings of the 17th International Symposium on Software Reliability Engineering, 62-74.

[11] Zimmermann, T., Premraj, R., Zeller, A. 2007. Predicting defects for eclipse, In Proceedings of the International Conference on Software Engineering, 9.

[12] Eclipse project development: Historical information about past releases, http://www.eclipse.org/eclipse/development/.

[13] Schröter, A., Zimmermann, T., Premraj, R. Zeller, A. 2006. If Your Bug Database Could Talk, In Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, 18-20.

[14] Nagappan, N., Murphy, B., Basili, V. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study, In Proceedings of the 30th International Conference on Software Engineering, 521-530.

[15] Boetticher, G., Menzies, T., Ostrand, T. 2007. Promise Repository of empirical software engineering data, http://promisedata.org/repository, Department of Computer Science.