Got Issues? Do New Features and Code Improvements Affect Defects?

Daryl Posnett
Department of Computer Science
University of California, Davis
Davis, CA
dpposnett@ucdavis.edu

Abram Hindle
Department of Computer Science
University of California, Davis
Davis, CA
ah@softwareprocess.es

Prem Devanbu
Department of Computer Science
University of California, Davis
Davis, CA
devanbu@ucdavis.edu

***Abstract*—There is a perception that when new features are added to a system that those added and modified parts of the source-code are more fault prone. Many have argued that new code and new features are defect prone due to immaturity, lack of testing, as well unstable requirements. Unfortunately most previous work does not investigate the link between a concrete requirement or new feature and the defects it causes, in particular the feature, the changed code and the subsequent defects are rarely investigated. In this paper we investigate the relationship between improvements, new features and defects recorded within an issue tracker. A manual case study is performed to validate the accuracy of these issue types. We combine defect issues and new feature issues with the code from version-control systems that introduces these features; we then explore the relationship of new features with the fault-proneness of their implementations. We describe properties and produce models of the relationship between new features and fault proneness, based on the analysis of issue trackers and version-control systems. We find, surprisingly, that neither improvements nor new features have any significant effect on later defect counts, when controlling for size and total number of changes.**

I. Introduction

Like locusts upon a field, bugs, defects, faults, errors, and failures plague software development. Software defects are costly and detrimental; as a result, they have long been the subject of anecdote, observation, experience and sometimes superstition. In this paper, we focus on some specific hypotheses, concerning different types of changes to code:

- Change is buggy: frequently changed code is often buggy.
- New features introduce bugs: new features are immature, and immature code has a greater chance of being buggy [1].
- Improvements to existing features reduces bugs: improvements to code quality should reduce defect proneness.
- Major improvements, on the other hand, can cause bugs: if we change too much code, we have too much defect prone immature code.

But are these beliefs well-founded? Is there empirical evidence to support a belief that feature additions and improvements can increase future defect-repair effort?

Much software engineering research is dedicated to the etiology of defects and the software features that uncover future defects. For example, Ostrand et al. found evaluated software and process metrics to find that change-prone code tended to be defect-prone as well [2].

However, some changes, are specifically made to fix defects; and so, in fact, we can expect these changes be associated with fewer *future* defects, while other types of changes may actually introduce new defects. In this paper we will address these beliefs using a particularly interesting data-set.

We will exploit both version control systems (Git and Subversion) and an issue tracker used by the *Apache Software Foundation* (ASF): the JIRA issue tracker. ASF repositories are useful because *commits are often linked* to JIRA issues.

We rely on the annotations provided by developers and issue reporters, as they manually annotate JIRA issues by ticket type, these ticket types include: *bug* (defects), *new feature* (code that adds new functionality), and *improvement* (code that improves code quality). These issue tickets and their relationships within the JIRA database are the central foci of our study. We manually inspect many of these issue tickets for accuracy and to learn the purpose of each kind of ticket.

We then used regression modeling to study the relationship between *new features* and *improvements* changes and defect-fixing activity. Specifically, we *improvements* issues and *new features* issues to the *defect-repair effort* in later releases using the version control system and the issue tracker. The main contributions described in this paper are:

- We find that new features and improvements are *negatively* correlated with defect-fixing activity in the same file, in the *same* release.
- Consistent with prior research, we find that code changes are strongly, and significantly associated with *future* defect fixing activity.
- However, we find that, in fact, new features and improvements are *not* correlated with *future* defect-fixing activity. New feature additions, in particular, show *no* relation with future defect-fixing activity.
- Upon further manual examination, we find that new features, in particular, in these projects are subject to careful review; thus, we find *substantive support for Linus' law* in our data [3].

We argue that these results are *actionable*. Contrary to conventional wisdom and prior research our results show that

Project	Description	Files	Packages	Commits	Issues	Defects	Improvements	New Features
James 2.3.0 - 3.0-M2	Mail Server	375-477	39-85	4467	996	281	413	83
Lucene 2.2.0 - 3.0.3	Text Search Library	541-1368	45-102	12384	8607	2093	3612	647
Wicket 1.3.0 - 1.3.7	Web Framework	1894-1947	246-249	7505	2935	953	477	156
XercesJ 2.8.1 - 2.11.0	Java XML parser	740-827	67-71	4934	576	307	189	80

TABLE I: Apache projects used in this study.

the *new features* and the *improvements* documented in the issue tracker do not interact with future defects. A qualitative deep-dive into the data suggests an explanation: a more rigorous inspection and vetting process for new feature code.

A. Issue Tracking

Issue trackers, like JIRA, are commonplace in OSS development. The JIRA issue tracking system supports multiple issue types. According to the JIRA documentation, a defect, labeled as *Bug* is "A problem which impairs or prevents the functions of the product", a *New Feature* is, "A new feature of the product", and an *Improvement* is, "An enhancement to an existing feature" [4]. Other kinds of custom issue types that are commonly used include *Tasks*, *Tests*, *SubTasks*, and *Wishes*. In this work we consider the relationship between the standard issue types of *Bugs*, *Improvements*, and *New Features*.

1) Bugs: There is a substantial body of research that links various code and process properties to the presence of defects [2], [5], [6], [7]. In particular, the size of code (measured as LOC) and how much it changes (measured either as the number of commits or the degree of "churn", *viz.*., added or change lines of code) have all been positively linked with the defect propensity of source code [8].

2) New Features: New features add new functionality to the code. New features would typically be in response to changing user requirements and may require both new code as well as significant changes to existing code. New features tend to imply new defect-prone code [1].

3) Improvements: Intuitively, *improvements* to code should improve the quality. Improvements are primarily perfective in nature, but still could induce defects.

Our first pair of research questions, which we address by doing a manual, qualitative case study, concerning the accuracy of the data:

Research Question 1: Are issues marked *new feature* actually new features?

Research Question 2: Are issues marked *improvements* actually improvements?

The next pair of research questions are concerned with the effect of new features and improvements on bug fixing activity. New feature code, being new functionality, can certainly be expected to increase fault-proneness, measured by proxy via bug fixing activity; we could also suspect that improvements to existing features, could also potentially introduce defects.

Research Question 3: Does adding new features to code affect defect fixing activity in the same release?

Research Question 4: Do improvements to code affect defect fixing activity in the same release?

Quality problems in code often manifest well-after the code is actually introduced. Our next pair of question address if the effect of code improvement on defects within the *same* release is not as strong as the effect on later releases.

Research Question 5: Do new features affect code quality in a future release?

Research Question 6: Do improvements to code affect code quality in a future release?

B. Motivation

In the field of defect prediction there have been numerous studies that consider structural and source code centric features [7]. While this obviously has merit, and has been quite successful, it is less common to see process-related data [8], [9], [6], such as issue tracker data that is strongly related to defect data. Thus, the effects of process-related behaviour, such as improvements or new features, on fault-proneness, have not been as well-studied.

II. Case Study

This case study is meant to ensure the reliability and construct validity of this study by investigating what the issue tracker data consists of and if its issue types of *bugs*, *new features* and *improvements*, are accurately applied. We manually read and tagged 240 issues. By doing this we could observe that performance, refactoring and documentation changes are associated with *improvements*.

Our methodology for the case study of the sampled JIRA issues used a manual coding approach, based on grounded theory. Our sampling approach considered a) all issues in general, and b) linked issues, issues linked to commits, in particular. Within the random sample, 21% of improvements

and new features were linked where 24.4% of all issues across 4 projects were linked. We sampled 10 random and 10 linked *new features*, *improvements* and *bugs* from each project (40 random and 40 linked issues per project). We inspected 240 issues in total; a subset of the issues used within the regression study.

We annotated each issue tags derived using a grounded theory approach. The final tags were *actual feature*, *actual improvement*, *actual bug*, *contribution* (a patch or content), *external contribution* (contributions from someone without commit access), *performance* (efficiency related), *refactor* (a refactoring), *unaccepted* (rejected contributions and reports), and *documentation*. The annotator (the second author) read the JIRA issues annotated the issue with one of these tags. These annotations were briefly inspected by the first author.

A. What are new features?

Figure 1a depicts the distribution of tags between projects, and features. A Persian language analyzer is an example of a new feature from Lucene, many other new features were backports from later versions. We investigated if the *new features* type was consistently applied:

> **Result 1:** *Across all the projects, most of the issues tagged* new features *are indeed new features. In fact* 79/80 *(*98.8%*)* new features *inspected were consistent. Although* 12/80 *of the new features could be bug fixes as well.*

New features issues added functionality or addressed new or existing requirements. 18/80 of the *new features* were from an *external contributor* and had received a code inspection.

B. What are improvements?

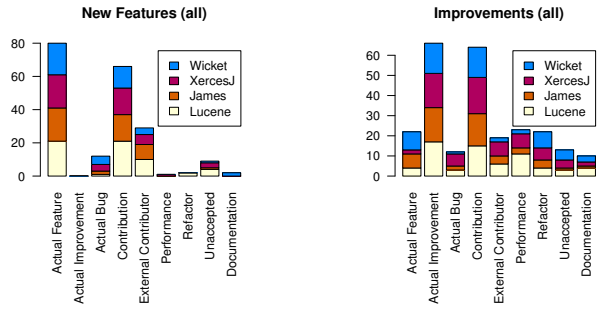
We wanted to ensure that the issue types of *improvements* were consistently applied:

> **Result 2:** *Across all projects the issues marked* improvements *were indeed usually describing improvements. Those that are not* improvements *are usually* new features *(*22/80*) and sometimes* bugs *(*3/80*). In fact most of the* improvements *inspected were consistent* 66/80 *(*82%*).*

Figure 1b depicts the relationship and distribution of tags between projects, and improvements. It clearly shows that many improvements are potentially bugs and some are really new features.

10/80 of the *improvements* inspected were documentation changes (Javadoc, websites, manuals, and examples).

Many of the *improvements* issues dealt with non-functional requirements such as maintainability (refactorings), performance and usability (UI changes). An example of an external improvement from Xerces was to change the access modifiers of a class to allow extension. 9/80 of *improvements* were code-inspected external contributions.



(a) (b)

Fig. 1: Distribution of improvements and new features by tag.

C. What are Bugs?

We inspected linked and unlinked *bug* issues and we found that 79/80 *bug* issues were in fact *bugs*. We also found that older projects such as Xerces had many *bugs* that were not linked to commits because JIRA was not in use at the time. Thus the *bug* issue type seems to be consistently applied.

D. How are JIRA issues used?

We observed that *new features*, *improvements* and *bugs* were consistently labelled as such. *Improvements* were rejected more by the project's developers than *new features* (13/80 versus 8/80).

JIRA issues were used by external contributors to submit and discuss code contributions with the core developers. 30% or 72/240 of the inspected issues were external contributions. 54% or 39/72 of external contribution tagged issues had a code inspection discussion. This supports our suggestion that *Linus's Law* is potentially negating the effect of the new features and improvements have on defects.

1) Linked Issues versus Population: In our random sample, only 21% of the *improvement* and *new feature* issues were actually linked to version control commits by a JIRA ID. Often backports were not linked in that particular issue.

Across linked and random samples *features* and *issues* received an equal amount of discussion. There were more actual bug report-like issues found in the linked sample. Linked samples more often accepted as contributions.

2) Issue Use per Project: James in particular had fewer unaccepted and documentation issues made against it. Xerces was more infrastructural and had many bug-fix oriented issues labelled as new features and improvements. Xerces did not have many linked new features because JIRA was adopted after Xerces was mature. Wicket had fewer external contributions of improvements and new features than others. Often Wicket's new features were bug fixes. Lucene had many improvements that were serious performance bugs (for example exponential time algorithms) labelled as improvements.

III. Regression Study

In the previous section we performed a manual case study on a sample of issues from the Jira database. In this section we

perform a regression study over the full dataset as presented in Table I. We begin by discussing the relationship between *improvements*, *new features*, and *bugs* in the same release. Any relationship will be dependent on overlap between bug fixing activity and the addition of improvements and new features. Table II shows that the ratio of files linked to new features and improvements in each project that were also linked to bug fixes varies but is non-zero. Consequently, we use *negative binomial regression* to regress the number of bug fixes within a release on the number of improvements and new features to asses the nature of the relationship between the activities. Negative binomial regression is an appropriate here because it can handle *overdispersion* in the response, which is almost always the case with software defect data [10].

	Features	Improvements
James	0.18	0.43
XercesJ	0.06	0.14
Lucene	0.17	0.58
Wicket	0.04	0.18

TABLE II: Fraction of files containing linked issues in files that are also linked to defects.

For these regression models we are primarily interested in the direction of the effect of improvement and new feature activity on bug fixing activity. We want to control as much as possible for other sources of variation that might be incorrectly attributed to the variables of interest. We control for size of the file by including the log transformed lines of code and we control for the number of changes to the file by including the square root of the number of commits. Comparison with the non-transformed variable shows that this transformation yields a better fit using Vuong's non-nested test with p-value < 0.0005 in all cases [11].

In addition we include dummy variables for each release to capture all *between* release variation for the model allowing us to focus on the *within* release contributions of the remaining independent variables on the outcome [10]. This approach has been used previously in modeling software outcomes [2], [12]. We include these factors purely for control, neither the significance nor the value of the coefficients are relevant to our results which are presented in Table III.

For all projects our controls of change activity and size are positively correlated with bug fixing activity as expected. Change and size are well known predictors of defects and we expect that larger files that change more often will have a greater number of defects, and consequently, higher defect fixing activity. We checked for multicollinearity by verifying that model *VIF (Variance Inflation Factor)* was within acceptable bounds [10]. With respect to our variables of interest, for Lucene, Xerces, and Wicket, improvements and new features were negatively associated with defect fixing activity. Adding new features to a file, or adding improvements to a file, in the same release is correlated with lower bug fixing activity in that release. We note here that, modulo developer time, activity over issues within a single file is not a zero-sum game, *viz.*,

	James	Lucene	Wicket	XercesJ
improvements	-0.0628	**-0.3965**	**-1.3167**	**-0.6779**
	(0.0763)	(0.0241)	(0.0730)	(0.1839)
new features	**0.3470**	**-0.3233**	**-0.8528**	**-0.6744**
	(0.1517)	(0.0538)	(0.1200)	(0.2481)
$log(\text{loc} + 0.5)$	**0.5758**	**0.1912**	**0.2290**	**0.4595**
	0.0754	(0.0288)	(0.0288)	(0.0585)
$\sqrt{\text{commits}}$	**0.5826**	**1.8346**	**2.7173**	**1.5610**
	0.0926	(0.0480)	(0.0749)	(0.0950)

(a) Number of Current Defects as Response

	James	Lucene	Wicket	XercesJ
previous improvements	-0.0458	-0.0515	-0.2723	0.1921
	(0.1829)	(0.0349)	(0.1216)	(0.1861)
previous new features	0.5518	-0.0380	-0.3383	-0.0275
	(0.2309)	(.0745)	(0.2518)	(0.3621)
$log(\text{previous loc} + 0.5)$	**1.3986**	**0.5036**	**0.8987**	**0.8697**
	(0.2537)	(0.0334)	(0.0518)	(0.0732)
$\sqrt{\text{previous commits}}$	0.3861	**0.7437**	**0.9060**	**0.2485**
	(0.2630)	(0.0604)	(0.1062)	(0.1062)

(b) Number of Current Defects as Response with features from previous release

TABLE III: Relevant coefficients for negative binomial regression models over files. Significant coefficients are bolded after correcting p-values using Benjamini Hochberg correction [13]. Standard errors are shown in parenthesis.

developer attention is variable. We can see from Table II that developers are performing multiple activities on the same file within a single release. The results of our models suggest, however, that the choice to improve existing code or add new features is less likely to co-occur with bug fixing activity in the same file. The results for James are somewhat different in that improvements are not significant and adding new features is positively associated with bug fixing activity, albeit, at a lower significance.

So with respect to our third and fourth research questions which address the relationship between code improvement and defect fixing activity:

Result 3: *Adding new features to code is negatively correlated with defect fixing activity, in the same file, for three out of four projects studied. In the fourth project adding new features is positively correlated with defect fixing activity.*

Result 4: *Improvements to code are negatively correlated with defect fixing activity in three out of four projects studied. Improvements were not significant in James.*

In other words, more often than not, developers choose to either address improvements and new features, or fix bugs. Only in James did we see a positive relationship between new feature activity and defect fixing activity.

We now turn to the central (title) question of the paper, concerning the *current* effect of *past* new feature and improvement effort. For each of the four projects we regressed bugs in the

current release on size and commits from the previous release as well as the number of new features and improvements from the previous release. The results of are presented in Table III. With respect to new features, the coefficients were not significant in any of the projects studied. This may, in part, be a consequence of the fact that many new features are external contributions which often require core members to inspect code prior to commit. Inspections can be effective at detecting defects [1]. Other have shown that a lack of significance can be important when trying to determine if a relationship or a lack of relationship exists between variables [14]. We used the same nested modeling approach used in [14]: we first built a model considering only the control variables of size and previous commit history; and then built the full model with previous new features and improvements to judge their explanatory power.

Result 5: *Adding new features does not directly contribute to later file fixing activity in any of the projects studied.*

We also find that previous code improvement does not affect defect fixing activity in later releases. For all projects the coefficient for improvements was not significant after p-value correction.

Result 6: *Adding improvements to code does not directly contribute to later file fixing activity in any of the projects studied.*

In all cases the models built with previous improvements and previous new features did not add statistically significant explanatory power to the models as determined by a chi-sqare test of nested model fit.

Discussion: Our findings have some implications. First, while they confirm prior results that changes *per se* do engender subsequent defect-repair effort, new features and improvements are *not* associated with defect-repair at the file level. Second, a manual inspection of the issues suggest a cause: new feature and improvements (especially the former) are subject to careful manual review, and thus may be benefiting from "many eyeballs" [3] effect. Finally, this approach suggests that future defect/effort prediction studies might well benefit from exploiting the available data embedded in issue histories.

IV. Conclusion

We have exploited the linked process data from the JIRA issue tracker and version control systems of multiple Apache projects in order to discuss the relationship between *improvements* and *new features* to the defect proneness of the source code that is changed.

Our case study confirmed that *improvements* and *new features* were in fact annotated correctly. Improvements were often refactorings, UI improvements and software quality relevant.

We then tried to model this relationship between *improvements*, *new features* and *defects* using a count regression model and the count of *improvement* and *new feature* issues as predictors. One would expect that *new features* would cause defect prone code while *improvements* would improve quality and reduce the defect-proneness of code. We observed, however, that in most cases, both file activity in *improvements* and *new features* is related to a reduction of activity in defect fixing in the same release. We also observed that code improvements and new features in previous releases have no significant impact on bug fixing activity in later releases This lack of significance could indicate that we are observing *Linus' Law*: the inspection of improvements and new features reduces their changes' effect on defect proneness.

Acknowledgments

All authors gratefully acknowledge support from the National Science Foundation, grant numbers 0964703 and 0613949. Devanbu gratefully acknowledges support from IBM Research and Microsoft Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] C. Jones, *Applied software measurement: assuring productivity and quality*. New York, NY, USA: McGraw-Hill, Inc., 1991.

[2] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 86–96, July 2004.

[3] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[4] Atlassian, "JIRA Concepts: What is an issue," http://confluence.atlassian.com/display/JIRA/What+is+an+Issue, 2010.

[5] T. Koponen and H. Lintula, "Are the changes induced by the defect reports in the open source software maintenance," in *Proc. of the 2006 Inter. Conf. on Soft. Eng. Research (SERP'06)*, 2006, pp. 429–435.

[6] J. Ratzinger, M. Pinzger, and H. Gall, "Eq-mine: Predicting short-term defects for software evolution," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, M. Dwyer and A. Lopes, Eds. Springer Berlin / Heidelberg, 2007, vol. 4422.

[7] T. Holschuh, M. Pauser, K. Herzig, T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects in SAP Java code: An experience report," in *International Conference on Software Engineering*, 2009.

[8] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 364–373.

[9] P. Zhang and A. Mockus, "On measurement and understanding of software development processes," 2003.

[10] J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.

[11] Q. Vuong, "Likelihood ratio tests for model selection and non-nested hypotheses," *Econometrica: Journal of the Econometric Society*, pp. 307–333, 1989.

[12] D. Posnett, C. Bird, and P. Dévanbu, "An Empirical Study on the Influence of Pattern Roles on Change-Proneness," *Empirical Software Engineering, An International Journal*, pp. 1–28, 2010.

[13] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[14] C. Bird, N. Nagappan, P. T. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista." in *ICSE'09*, 2009, pp. 518–528.