Rapid Releases and Patch Backouts: A Software Analytics Approach

Rodrigo Souza
Federal University of Bahia, Brazil

Christina Chavez
Federal University of Bahia, Brazil

Roberto A. Bittencourt
State University of Feira de Santana, Brazil

Abstract

Mozilla's decision to release a new version of its products every 6 weeks (instead of every year) had profound impacts on users and developers, and was accompanied by significant changes in the release process. Were such changes enough to allow Mozilla to move faster without breaking things? What lessons can be learned from Mozilla's adoption of rapid releases? To answer these questions, we analyzed tens of thousands of commits and bug reports from Mozilla Firefox and talked to its developers. We show that, because of integration repositories, build sheriffs, and better testing tools, broken patches are being backed out – i.e., reverted – earlier, rendering the release process more stable.

Keywords: release engineering, rapid releases, software analytics, bug reopening.

1 Introduction

Release engineering deals with decisions that impact the daily lives of developers, testers, and users, and, thus, contribute to the success of a product. While gut feeling plays an important role in such decisions, it is increasingly important to leverage existing data, such as bug reports, source code changes, code reviews, build and test results, both to support decisions and to help evaluate current practices. The approach of exploring software engineering data to obtain insightful information is called *software analytics* [1].

The Mozilla Foundation recently changed its release process in a fundamental way: it moved from traditional, 12–18 month releases, to rapid, 6-week releases. The change was motivated by the need to deliver new features earlier to their products' users, keeping pace with the evolution of web standards, with the competition in the web browser market, and with the emergence of mobile platforms.

The impacts of Mozilla's adoption of rapid releases have been previously studied by researchers using the software analytics approach (see the sidebar "Previous Studies about Rapid Releases at Mozilla"). While previous studies focused on changes from the viewpoint of users, plug-in developers, and quality engineers, we focus on the impact of rapid releases on code integration issues, essential for the timely release of new versions. Particularly, we analyze how the backout rate evolved during Mozilla's process change.

Backout is the act of reverting a patch that was committed to a source code repository, either because it broke the build or, generally, because some problem was found in the patch. Backout implies rework, since it requires a new patch to be written, reviewed, and tested. A high backout rate is an indicator of a unstable process.

By analyzing tens of thousands of commits and bug reports from the Mozilla Firefox project, we found that the proportion of early backouts, i.e., backouts performed during the first integration build of a commit, increased, while late backouts dropped. This phenomenon can be explained by improvements in testing tools and code integration practices. This is a positive trend, since it means that problems are discovered earlier and, thus, less problems are expected during a release.

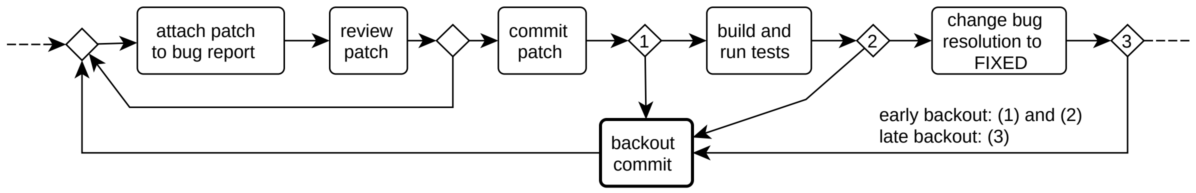
attach patch to bug report
review patch
commit patch
1
build and run tests
2
change bug resolution to FIXED
3
backout commit
early backout: (1) and (2)
late backout: (3)

Figure 1: Mozilla bug fixing process

2 Code Integration at Mozilla

In the last 5 years, the development process at Mozilla in general, and Firefox in particular, has been characterized by an intense application of code review practices and automated testing in multiple levels, such as unit testing and user interface testing. This process has been supported by tools such as Bugzilla, a bug tracking system, and Mercurial, a distributed version control system. This section starts describing the process as it was before 2011, and then explaining the changes that occurred after. Since we only analyze the period between 2009 and 2013, specifics of the process before 2009 and after 2013 are ignored.

2.1 Before 2011: Traditional Releases

Before March 2011, Firefox had been developed according to a traditional release schedule: features for the upcoming version were developed along with bug fixes and minor updates for the current stable release. Major features would only be delivered to users with the release of a major version, which occurred when planned features were implemented and tested. In practice, a new major version used to take from 12 to 18 months to be released [5].

Figure 1 summarizes the bug fixing process at Mozilla. A source code change is first proposed as a patch on Bugzilla. The developer proposing the change then requests a code review from another developer, which can either approve the patch or reject it, in which case a new patch should be written and reviewed. Once the patch is approved, the developer can commit it to the code repository.

All developers with commit access should commit to and pull changes from the Mozilla central repository, often abbreviated m-c. Nightly builds are created from this repository, so developers, testers and other stakeholders can test the most recent changes. Automated tests are run during the build process.

Mozilla central must be fairly stable, since it is the starting point for the development of new features and bug fixes. If the code in m-c fails to compile or breaks major features, it prevents testing of new changes. In this case, the offending commit

Previous Studies about Rapid Releases at Mozilla

Researchers have been studying the impacts of Mozilla's move to rapid releases under multiple perspectives. While the advantages of releasing features earlier are clear, Christian Plewnia and his colleagues showed that, in the first years of the change, the reputation of Firefox was harmed [2]. Some reasons include users being prompted to update the software with higher frequency and plug-in developers fearing that new releases would break the API. The reputation was since then regained with the implementation of silent updates and the introduction of extended support releases.
Regarding the impacts on the development itself, Mika Mäntylä and his colleagues showed that Mozilla had to hire more testers and narrow the scope of testing, since rapid releases did not leave enough time to run all manual tests at every release [3]. This approach seems to be paying off, because the number of post-release bugs have not changed significantly after the organization moved to rapid releases, as showed by Foutse Khomh and his colleagues [4].

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

needs to be backed out to stabilize the repository, or even fixed right away with another commit. In some cases the repository is closed to prevent further changes while stabilization is in process.

To prevent m-c from breaking frequently, developers can, in addition to running tests in their development machines, submit their patches to the so-called Try server before committing them. The Try server checks out a copy of m-c, applies the patch, builds the code, and runs automated tests. As it may take hours to build all platforms and run all tests, developers can choose to build a subset of the platforms and run a subset of the tests.

When developers are confident about their patches, they commit them to m-c. They need to wait for the next build cycle and watch the build to make sure the changes did not break the build or caused test failures. If any problem happens, they are responsible for backing out the bug. For this reason, developers are recommended to commit only if they are available in the next four hours [6].

Once the change is in m-c and tests pass, the corresponding bug report is updated with status RESOLVED and resolution FIXED. If any problem is detected in further tests, the commit is backed out and the bug report status is changed to REOPENED so it can be resolved again.

2.2 Since 2011: Rapid Releases

In March 2011, Mozilla started to release its products in short, 6-week, release cycles, beginning with the development of Firefox 5. In this release cycle, though, the process was still being stabilized. It was only in June that integration repositories, such as Mozilla inbound (m-i), were created. Thereafter, patches started to be committed to m-i, tested on m-i, and then merged once a day to m-c. In this scenario, patches that break m-i are backed out prior to merging.

One essential difference is that, instead of each developer watching the build and backing out their own commits, this job is now made by build engineers that take turns as *sheriffs*. Breaking the build on m-i is less of a problem in this case, since sheriffs back out troublesome patches before merging the code into m-c, which becomes more stable. Sheriffs are also responsible for changing the status of bug reports to RESOLVED/FIXED after patches are tested and merged.

3 Software Analytics Approach

To investigate how the backout rate changed when Firefox transitioned to rapid releases, we collected, transformed, and analyzed publicly available data produced by Mozilla engineers.

3.1 The Data

We used two primary sources of information: commit logs and bug reports. The commit logs were extracted from the central repository using the command `hg log`. Bug reports were made available as a SQL database dump by a Mozilla engineer.

We also obtained release dates from Mozilla's wiki. We decided to analyze the development of versions 3.6 and 4.0, both developed under traditional release cycles, and versions 5 up to 27, developed under rapid, 6-week cycles. The development of versions 3.6 up to 27 amounts to more than four years of data, as shown in Figure 2.

The data was split into three periods: traditional releases, rapid releases (transitional), and rapid releases with integration repositories. The development of version 5 was isolated in the transitional period because it was atypical: the release cycle was longer and integration repositories did not exist back then. Each bug report was mapped to a release according to the date of its first bug fix commit.

3.2 Mapping Commits to Bug Reports

Commits were classified as bug fixes or backouts, and mapped to the bugs they fix or revert. To this end, we relied on conventions developers use when writing commit messages.

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

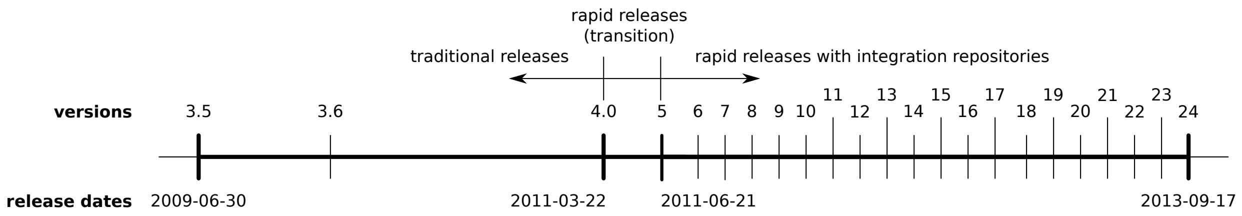

Figure 2: Firefox release history and periods being studied

Bug fixes. Bug fix commits start with the word "bug" followed by a 5- to 6-digit number that uniquely identifies the bug. Example: "Bug 939080 - Allow support-files in manifests to exist in parent paths; r=ted".

Backouts. Backout commits contain the expression "back out" or a variation, such as "backout", "backs out", "backed out", and "backing out", followed by either a 7- to 12-digit hexadecimal number, referring to the bug fix commit being backed out, or the number of the bug whose fix it backs out, or both. Example: "Back out 7273dbeaeb88 (bug 157846) for mochitest and reftest bustage".

3.3 Early and Late Backouts

For analytical purposes, each backout was further classified as early, if it occurred before the corresponding bug report had its resolution changed to FIXED, or late, if it occurred after that. In practical terms, an *early backout* occurs when a commit breaks its first build, usually either because it prevented the code from compiling, or because it made automated tests fail. If, on the other hand, a problem is only discovered afterwards, then a *late backout* is performed. A bug fix can even be backed out more than once, both early and late.

3.4 Data Analysis

Having collected the raw data, mapped commits to bug reports, and classified backouts, we could determine, for each bug, whether it was ever backed out and, if so, whether it was an early or a late backout. Backout rates were computed as the ratio between the number of bug reports associated with at least one backout commit and the number of bugs associated with at least one bug fix commit.

First, we plotted monthly backout rates. Then, we computed backout-related metrics of the three periods, and applied statistical tests (such as Fisher's exact test and Wilcoxon signed-rank test) to evaluate whether the differences were statistically significant. Finally, we contacted Firefox engineers using the `firefox-dev` mailing list. We reported our numbers and asked them to explain the results according to their experience.

4 Results

In the following subsections, we report the evolution of backout rate and other metrics, explain why they have changed over time, and analyze how they impacted Firefox users and developers.

4.1 The Numbers

Table 1 shows the metrics computed for the three periods. First of all, the number of bug fixes per day almost doubled under rapid releases. Upon further inspection, we can see that this increase is highly correlated with a growth in the number of regular committers; thus, we can infer that the developer workload have not changed significantly in the period. As a Mozilla engineer stated, "a developer can only do so much work; growth is mostly adding developers nowadays, not the individual doing more".

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

Table 1: Metrics per release model

	traditional	**rapid (transitional)**	**rapid (with integration)**
Number of bug fixes	11,220	1,893	30,085
Number of days	631	90	892
Bug fixes per day (average)	17.8	21.1	33.7
Number of committers*	45.5	64.6	92.7
Number of fixes backed out	702	173	2831
Proportion of fixes backed out	6.3%	9.1%	9.4%
Bugs backed out per day (average)	1.1	1.9	3.2
Early backout rate	3.5%	5.1%	8.3%
Late backout rate	3.1%	4.9%	1.5%
Proportion of early backouts**	56.7%	55.5%	87.7%
Median time-to-backout (hours)	5.7	12.6	4.2

* Within each month, we counted the number of developers with at least 5 commits and then averaged across the months in each period. ** Proportion of backed out bug fixes that were backed out early.

The numbers also show that backout is a relevant problem. Under rapid releases, 9.4% of all bugs that were fixed eventually got backed out, which represents 3.2 bugs backed out every day, on average.

Figure 3 shows that the overall backout rate increased under rapid releases. A few Mozilla engineers pointed out that the backout rate may have been underestimated under traditional releases, because the backout culture has become more prevalent after the introduction of sheriff-managed integration repositories. Before that, it was more common to fix a broken commit by committing again, without explicitly backing out the first commit:

> Where bugs might have had broken patches land and gotten fixed in-tree, our current process and tree sheriffs will backout obvious failures until the bugs get fixed before landing.
>
> [With inbound and sheriffs,] we do not end up fixing the issues with followup after followup fix but rather have them backed out right away.

In order to better understand the impacts of backouts, though, we broke them down into early and late backouts. Figure 4 shows that, while the early backout rate shows a growing trend, the late backout rate dropped significantly after the introduction of integration repositories.

The numbers in Table 1 reinforce that there was a shift towards earlier problem detection. Among all backouts, the proportion of early backouts increased from 57% to 88%. Time-to-backout, i.e., the time it takes for an inappropriate bug fix to be reverted, also dropped after the adoption of rapid releases.

4.2 What Does It All Mean?

The numbers show that overall and early backout rate may have increased, while late backouts became less frequent. But what do these trends reveal about changes in Firefox's process and context? Asked about it, eight Mozilla engineers offered explanations.

Larger code base and number of products. Some engineers explained the increase in overall backout rate suggesting that the code base grew over time and, as result, code conflicts became more likely. The number of supported platforms also increased, since Firefox needs to support both new platforms, such as Windows 8, and older ones, such as Windows XP. Also, new products emerged, such as Firefox for Android and Firefox OS, that share code with the desktop web browser. As explained by an engineer:

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

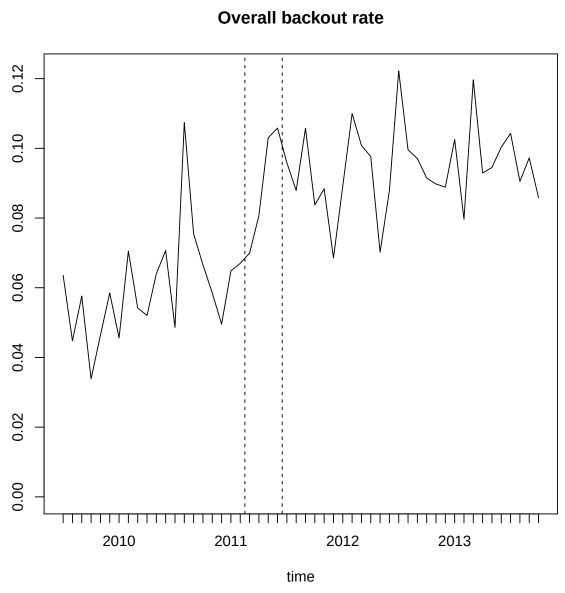

Figure 3: Overall backout rate over time. Dashed vertical lines represent important events: the left one, the start of the first rapid release cycle; the right one, the introduction of integration repositories.

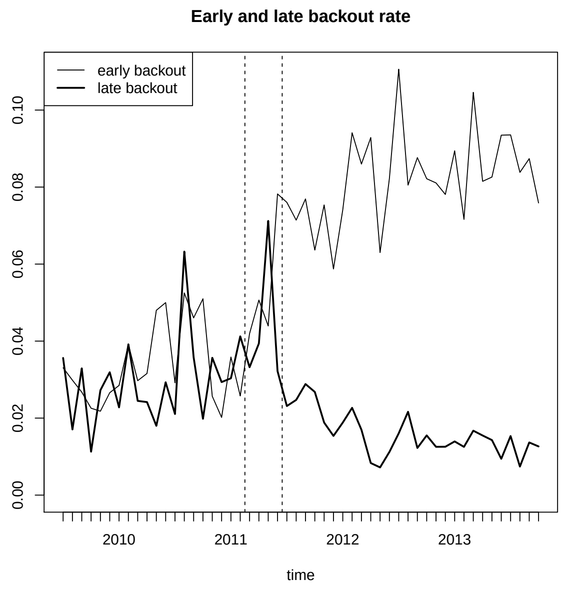

Figure 4: Early and late backout rate over time. Dashed vertical lines represent important events: the left one, the start of the first rapid release cycle; the right one, the introduction of integration repositories.

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

> We have a lot more stuff that can break, on more platforms, as well as more tests — these days we don't have everyone working on just Firefox. Code landing for B2G can break Fennec, for example, and B2G devs don't build and test on Fennec locally. Those kinds of changes will be caught and backed out when they hit the trees, not found beforehand. [Note: "B2G" refers to Firefox OS, "Fennec" refers to Firefox for Android, and "tree" refers to a code repository, such as inbound.]

Evolution of testing tools. The increasing early and decreasing late backout rate can be partially explained by the evolution of the automated testing tool set. According to Mozilla engineers, the emergence of better testing tools have contributed to improve earlier detection of problems, and even to detect problems that would otherwise go unnoticed, such as hard-to-detect memory leaks:

> ... our automated testing has improved considerably since [release] 3.5. A number of memory leak finding tools have been integrated into our test environments that are improving our early catch-rate.

Integration repositories and backout culture. The growing early backout rate can be explained by the creation of sheriff-managed integration repositories and their effect on the way developers test their code. Before 2011, developers used to push changes directly to the central repository; therefore the changes needed to be thoroughly tested to avoid breaking the builds or introducing bugs. Since June, 2011, developers started to commit to integration repositories, such as inbound, where tests are run and problematic patches are backed out by the sheriff before merging changes to central, thus keeping it stable. As a result, developers were encouraged to commit to inbound having performed less testing. As stated in Mozilla's wiki:

> Breaking it [the integration repository] rarely is ok. Never breaking the tree [the integration repository] means you're running too many tests before landing [committing to the repository].

Two Mozilla engineers reinforced this view:

> (...) in the 'old days', you were expected to have built, tested, done a Try build, etc. before the patch landed.
>
> (...) the backout aggressiveness was even explicitly mentioned when we switched.

4.3 So What?

What do the changes in overall, early and late backout rate mean to Firefox users and developers? To answer this question, we first have to understand the impacts of early and late backouts.

Impact on developers. Every backout induces rework by requiring the development of a new, improved, patch. However, the increase of early backouts, in Mozilla's case, does not seem to cause overhead. Instead, it reflects a cultural change towards committing patches before testing them comprehensively, therefore reducing the effort needed to test patches. Such change is only possible because broken patches no longer reach the central repository: they are now committed to a integration repository and backed out by sheriffs before merging. Sheriffs also make sure that patches that break the build are backed out as soon as possible, reducing the amount of time in which the repository need to closed:

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE

> I'd say amount of time spent testing patches before landing, and amount of time wasted with trees [i.e., repositories] closed due to bustage [i.e., broken build], were reduced.

Although all backouts induce rework, late backouts are more severe. First, the longer it takes for a fix to be backed out, the more time developers spend trying to remember the context and set up their environments to create an improved patch. Also, problems that are not resolved early may end up in a release; hence, users may have to wait another release cycle to have the definitive bug fix delivered to them. Finally, in the presence of integration repositories, inappropriate commits that are not backed out early end up in the central repository, on which developers base their work; by the time the commit is backed out, there may be many other commits that depend on it.

Therefore, since there was a shift towards earlier backouts, we can infer that the creation of sheriff-managed integration branches reduced the effort required to integrate bug fixes.

Impact on users. While Mozilla's move to rapid releases was a success from the release engineering perspective, it upset users because of frequent update notifications and broken plug-in compatibility. As summarized by the then chair of Mozilla Foundation on his blog post [7]:

> We focused well on being able to deliver user and developer benefits on a much faster pace. But we didn't focus so effectively on making sure all aspects of the product and ecosystem were ready.

Regarding the impact of backouts, though, user quality perception has not changed. After being committed to the central repository, a bug fix goes through two other repositories, called aurora and beta, where more tests are performed during two release cycles before it is released to the general public. Thus, only very late backouts would affect users, and very late backouts are rare under both traditional and rapid releases. As stated by a Mozilla engineer:

> I think our development process gives us a margin of safety to detect regressions well before the code actually reach the hands of users. As the user base of each repository grows gradually, we have an effective way to detect unexpected problems well in advance.

5 Conclusion

Given the repercussions and potential impacts of Mozilla's decision to change its release process, from traditional to rapid releases, we decided to investigate how the backout rate evolved during this change using a software analytics approach. A naive analysis of the numbers would suggest that the adoption of rapid releases resulted in a less stable process, with bug fixes being more frequently backed out. However, further inspection showed that, under rapid releases, there was a shift towards early backouts, which is beneficial, since late backouts have worse impacts on both developers and users.

By computing metrics and talking to Mozilla engineers, we were able to identify two concrete measures taken by Mozilla that helped in keeping the process stable while allowing it to move faster:

- **Improve automated testing tools**. Automated testing is used to detect problems very early in the process. The earlier problems are detected, the faster it is to fix them and the earlier the fixes can be delivered to end users.
- **Use integration repositories**. It is ok to break the product more often, as long as it does not affect other developers' work. Integration repositories allow developers to focus on writing code, while sheriffs

ensure that bad quality code is filtered out as soon as possible, before reaching the main source code repository.

While these two measures can be used to improve any project, the overhead involved in putting them into practice is more justifiable under rapid releases, since in this context it is important to keep the source code stable as often as possible. Having frequently stable code is also important when Mozilla has to deliver a "chemspill" release, i.e., a release that fixes critical security issues and, thus, should reach users as soon as possible.

Our data analysis uncovered previously unknown information about the evolution of early and late backouts in Firefox and, therefore, helped evaluate the impacts of the adoption of rapid releases by Mozilla. In the future, such analysis could be integrated into an analytics tool constantly updated with backout rates and other metrics. That way, release engineers would have easier access to up to date information about the process, enabling them to evaluate the impact of yet-to-be-made decisions.

Acknowledgement

The authors would like to thank all Mozilla engineers who provided feedback on the results.

References

[1] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, and T. Xie, "Software analytics in practice," *IEEE Software*, vol. 30, no. 5, pp. 30–37, 2013.

[2] C. Plewnia, A. Dyck, and H. Lichter, "On the influence of release engineering on software reputation," in *2nd International Workshop on Release Engineering*, 2014.

[3] M. Mantyla, F. Khomh, B. Adams, E. Engstrom, and K. Petersen, "On rapid releases and software testing," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, Sept 2013, pp. 20–29.

[4] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? An empirical case study of Mozilla Firefox," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, June 2012, pp. 179–188.

[5] Mozilla, "Mozillawiki: Releases," 2014, https://wiki.mozilla.org/Releases.

[6] ——, "Committing rules and responsibilities," https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Committing_Rules_and_Responsibilities.

[7] M. Baker, "Rapid release follow-up," http://blog.lizardwrangler.com/?p=2996.

About the Authors

Rodrigo Rocha Gomes e Souza is a PhD student in the Department of Computer Science at the Federal University of Bahia. His research interests include empirical software engineering, release engineering, software evolution, and mining software repositories. Rodrigo holds an MSc in Computer Science from the Federal University of Campina Grande. Contact him at rodrigo@dcc.ufba.br.

Christina von Flach Garcia Chavez is a professor in the Department of Computer Science at the Federal University of Bahia. Her research interests include software design and evolution, and software engineering education. Christina has a PhD in Computer Science from the Pontifical Catholic University of Rio de Janeiro. She's a member of the ACM and the IEEE. Contact her at flach@ufba.br.

Roberto Almeida Bittencourt is an assistant professor at the State University of Feira de Santana. His research interests include software evolution and design, computing education, and CSCW. Roberto has a PhD in Computer Science from the Federal University of Campina Grande, with a research internship at the University of British Columbia. Contact him at roberto@uefs.br.

Digital Object Indentifier 10.1109/MS.2015.30 0740-7459/$26.00 2015 IEEE