

FOCUS: RELEASE ENGINEERING

Rapid Releases and Patch Backouts

A Software Analytics Approach

Rodrigo Souza and Christina Chavez, Federal University of Bahia

Roberto A. Bittencourt, State University of Feira de Santana

// *To investigate the results of Mozilla's adoption of rapid releases, researchers analyzed Firefox commits and bug reports and talked to Firefox's developers. The results show that developers are backing out broken patches earlier, rendering the release process more stable.* //



RELEASE ENGINEERING deals with decisions that impact the daily lives of developers, testers, and users and thus contribute to a product's success. Although gut feeling is important in such decisions, it's increasingly important to leverage existing data, such as bug reports, source code changes, code reviews, and test results, both to support decisions and to help evaluate current practices. The exploration of

software engineering data to obtain insightful information is called *software analytics*.¹

In 2011, the Mozilla Foundation fundamentally changed its release process, moving from traditional 12- to 18-month releases to rapid, six-week releases. The motivation was the need to deliver new features earlier to users, keeping pace with the evolution of Web standards, the competition among Web

browsers, and the emergence of mobile platforms.

Researchers have used software analytics to study the impact of Mozilla's adoption of rapid releases (see the sidebar). Those studies focused on changes from the viewpoint of users, plug-in developers, and quality engineers. Here, we focus on how rapid releases affect code integration, which is essential for the timely release of new versions.

In particular, we analyze how the *backout* rate evolved during Mozilla's process change. A backout reverts 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 because it requires writing, reviewing, and testing a new patch. A high backout rate indicates an unstable process.

Code Integration at Mozilla

Over the last five years, development at Mozilla in general, and Firefox in particular, has intensely applied code review and automated testing at 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. Here we describe the process before 2011 and the changes that occurred after. Because we analyze only the period between 2009 and 2013, we ignore specifics of the process before 2009 and after 2013.

Before 2011: Traditional Releases

Before March 2011, Firefox development followed a traditional release schedule. Features for the upcoming version were developed along with bug fixes and minor updates for the

FOCUS: RELEASE ENGINEERING

 PREVIOUS STUDIES
OF RAPID RELEASES AT MOZILLA

Researchers have been studying the impact of Mozilla's move to rapid releases under multiple perspectives. Although the advantages of releasing features earlier are clear, Christian Plewnia and his colleagues showed that, in the first years of the change, Firefox's reputation was harmed.¹ Some reasons include users being prompted to update the software more often and plug-in developers fearing that new releases would break the API. Since then, Firefox has regained its reputation by implementing silent updates and introducing extended support releases.

Regarding the impact on the development itself, Mika Mäntylä and his colleagues showed that Mozilla had to hire more testers and narrow the testing's scope because rapid releases didn't leave enough time to run all manual tests at

every release.² This approach seems to be paying off. The number of postrelease bugs hasn't changed significantly after Mozilla moved to rapid releases, as Foutse Khomh and his colleagues showed.³

References

1. C. Plewnia, A. Dyck, and H. Lichter, "On the Influence of Release Engineering on Software Reputation," presented at 2nd Int'l Workshop Release Eng., 2014; http://releng.polymtl.ca/RELENG2014/html/proceedings/releeng2014_submission_3.pdf.
2. M. Mäntylä et al., "On Rapid Releases and Software Testing," *Proc. 29th IEEE Int'l Conf. Software Maintenance (ICSM 13)*, 2013, pp. 20–29.
3. F. Khomh et al., "Do Faster Releases Improve Software Quality? An Empirical Case Study of Mozilla Firefox," *Proc. 9th IEEE Working Conf. Mining Software Repositories (MSR 12)*, 2012, pp. 179–188.

current stable release. Major features would be delivered to users only with the release of a major version, which occurred when planned features were implemented and tested. In practice, a new major version took from 12 to 18 months to be released.²

Figure 1 summarizes bug fixing at Mozilla at that time. A developer first proposed a source code change as a patch on Bugzilla. That developer then requested a code review from another developer, who approved or rejected the patch. In the latter case, a new patch was written and reviewed. Once the patch was approved, the developer committed it to the code repository.

All developers with commit access committed to and pulled changes from the Mozilla central repository, often abbreviated m-c. Nightly builds were created from m-c so that developers, testers, and other stakeholders could test the most recent changes. Automated tests ran during the build process.

The central repository had to be fairly stable because it was the starting point for developing new features and bug fixes. If the code in m-c failed to compile or broke major features, it prevented testing of new changes. In this case, the developer had to back out the offending commit to stabilize the repository or even fix it right away with another commit. In some cases, the repository was closed to prevent further changes while stabilization occurred.

To prevent m-c from breaking frequently, developers could, besides running tests in their development machines, submit their patches to the Try server before committing them. The Try server checked out a copy of m-c, applied the patch, built the code, and ran automated tests. Because building all the platforms and running all the tests might take hours, developers could choose to build a subset of the platforms and run a subset of the tests.

When developers were confident about their patches, they committed them to m-c. They had to wait for the next build cycle and watch the build to ensure the changes didn't break the build or cause test failures. If a problem happened, they had to back out the bug. So, developers were recommended to commit only if they were available for the next four hours.³

Once the change was in m-c and tests passed, the corresponding bug report was updated with the status Resolved and resolution Fixed. If further tests detected a problem, the commit was backed out, and the bug report status changed to Reopened so that it could be resolved again.

2011–2013: Rapid Releases

In March 2011, Mozilla started the six-week release cycles, beginning with the development of Firefox 5. In this cycle, though, Mozilla was still stabilizing the process. Only in June did it create integration repositories, such as Mozilla inbound (m-i).

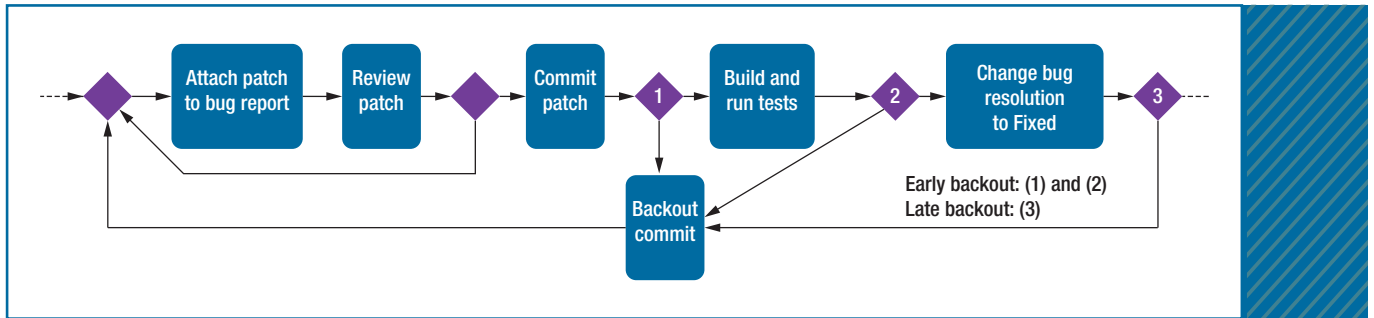


FIGURE 1. Mozilla's bug-fixing process between 2009 and 2013.

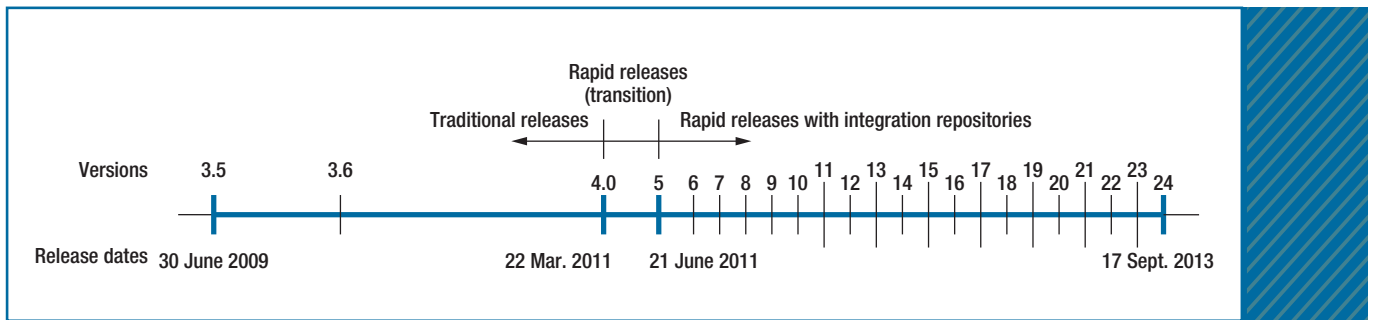


FIGURE 2. Firefox's release history for the periods being studied. The development of version 5 was isolated in the transition period because it was atypical: the release cycle was longer and integration repositories didn't exist then.

Thereafter, patches were committed to m-i, tested on m-i, and then merged once a day with m-c. Patches that broke m-i were backed out before merging.

Instead of each developer watching the build and backing out his or her own commits, build engineers took turns as *sheriffs* who performed this job. Breaking the build on m-i was less of a problem because sheriffs backed out troublesome patches before merging the code into m-c, making it more stable. Sheriffs also changed the bug report status to Resolved, with resolution Fixed, after they tested and merged the patches.

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.

The Data

We used two primary information sources: commit logs and bug reports. We extracted the logs from m-c using the command `hg log`. A Mozilla engineer made the bug reports available as an SQL database dump.

We also obtained release dates from Mozilla's wiki. We analyzed the development of versions 3.6 and 4.0, both developed under traditional release cycles, and versions 5 through 27, developed under six-week cycles. The development of versions 3.6 to 27 amounted to more than four years of data (see Figure 2).

We split the data into three periods: traditional releases, transitional rapid releases, and rapid releases

with integration repositories. The development of version 5 was isolated in the transition period because it was atypical: the release cycle was longer, and integration repositories didn't exist then. We mapped each bug report to a release according to the date of its first bug fix commit.

Mapping Commits to Bug Reports

We classified commits as bug fixes or backouts and mapped them to the bugs they fixed or reverted. To this end, we relied on conventions developers use when writing commit messages.

Bug fixes. Bug fix commits start with the word “bug” followed by a five- to six-digit number uniquely identifying the bug. An example is, “Bug 939080 - Allow support-files in manifests to exist in parent paths; r=ted.”

FOCUS: RELEASE ENGINEERING

TABLE 1

Metrics per release model.

Metric	Release model		
	Traditional	Transitional rapid	Rapid with integration
No. of bug fixes	11,220	1,893	30,085
No. of days	631	90	892
Avg. no. of bug fixes per day	17.8	21.1	33.7
No. of committers*	45.5	64.6	92.7
No. of fixes backed out	702	173	2,831
Fixes backed out (%)	6.3	9.1	9.4
Avg. no. of bugs backed out per day	1.1	1.9	3.2
Early backout rate (%)	3.5	5.1	8.3
Late backout rate (%)	3.1	4.9	1.5
Fixes backed out early (%)†	56.7	55.5	87.7
Median time-to-backout (hrs.)	5.7	12.6	4.2

* Each month, we counted the number of developers with at least five commits; we then averaged this number across the months in each period.

† The proportion of backed-out bug fixes that were backed out early.

Backouts. Backout commits contain the expression “back out” or a variation, such as “backout,” “backs out,” “backed out,” or “backing out,” followed by 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. An example is, “Back out 7273dbeaeb88 (bug 157846) for mochitest and reftest bustage.”

Early and Late Backouts

We classified a backout as early if it occurred before the resolution of its bug report changed to Fixed. We classified the backout as late if it occurred after that. In practical terms, an early backout occurs when a commit breaks its first build, usually because it either prevented the code from compiling or made automated tests fail. If a problem is discovered only afterward, a late backout

is performed. A bug fix can even be backed out more than once, both early and late.

Data Analysis

After collecting the data, mapping commits to bug reports, and classifying backouts, we determined whether each bug was ever backed out and, if so, whether the backout was early or late. We computed the backout rate as the ratio of the number of bug reports associated with at least one backout commit to the number of bugs associated with at least one bug fix commit.

First, we plotted monthly backout rates. Then, we computed backout-related metrics for the three periods. We applied statistical tests (such as Fisher’s exact test and the 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.

The Results

Here we report on the evolution of backout rate and other metrics, explain why they changed over time, and analyze how they affected Firefox developers and users.

The Numbers

Table 1 shows the metrics for the three periods. First, the number of bug fixes per day almost doubled under rapid releases. This increase is highly correlated with a growth in the number of regular committers. So, we can infer that the developer workload didn’t change significantly over 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.

The numbers also show that backout was a relevant problem. Under rapid releases, 9.4 percent of all bugs that were fixed eventually got backed out, an average of 3.2 bugs daily.

Figure 3 shows that the overall backout rate increased under rapid releases. A few Mozilla engineers pointed out that the backout rate might have been underestimated under traditional releases because the backout culture became more prevalent after the introduction of sheriff-managed integration repositories. Before that, developers usually fixed a broken commit by recommitting, 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 back-out obvious failures until the bugs get fixed before landing.

[With inbound and sheriffs,] we do not end up fixing the issues with follow-up after follow-up fix but rather have them backed out right away.

To better understand the backouts' impact, we broke them down into early and late backouts. Figure 4 shows that, although the early-backout rate grew, the late-backout rate dropped significantly after the introduction of integration repositories.

Table 1 reinforces that a shift occurred toward earlier problem detection. Among all backouts, the proportion of early backouts increased from 57 to 88 percent. The

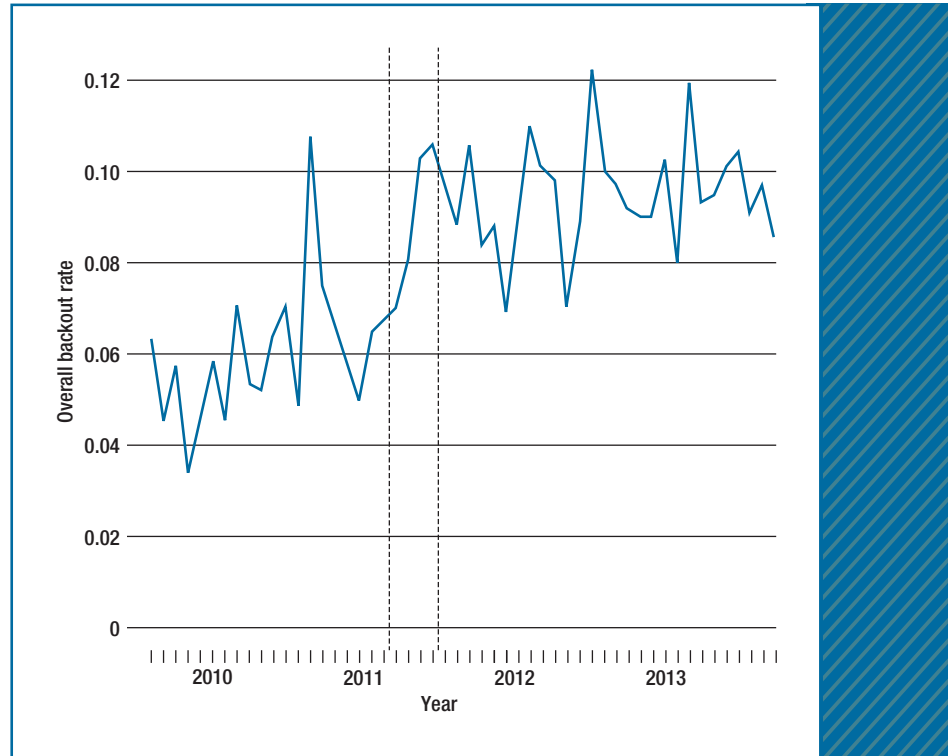


FIGURE 3. The overall backout rate over time. The dashed vertical lines represent important events. The left one is the start of the first rapid-release cycle; the right one is the introduction of integration repositories. The overall backout rate increased under rapid releases.

time-to-backout (the time for an inappropriate bug fix to be reverted) also dropped after the adoption of rapid releases.

What Does It All Mean?

What do the backout trends reveal about changes in Firefox's process and context? Eight Mozilla engineers offered explanations.

A larger code base and more products. Some engineers explained the increase in the overall backout rate by suggesting that because the code base grew over time, code conflicts became more likely. The number of supported platforms also increased because Firefox must 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 one engineer explained,

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 [the Firefox OS] can break Fennec [Firefox for Android], 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 [code repositories], not found beforehand.

FOCUS: RELEASE ENGINEERING

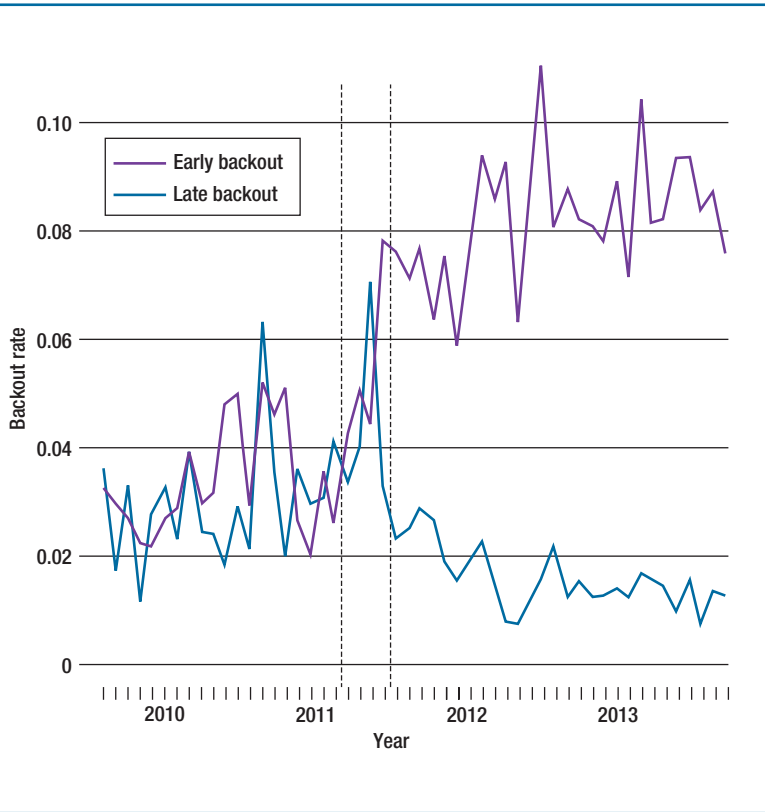


FIGURE 4. The early-backout and late-backout rates over time. The dashed vertical lines represent important events. The left one is the start of the first rapid-release cycle; the right one is the introduction of integration repositories. Although the early-backout rate grew, the late-backout rate dropped significantly after the introduction of integration repositories.

The evolution of testing tools. The increasing early-backout rate and decreasing late-backout rate were due partly to the evolution of the automated testing toolset. According to Mozilla engineers, the emergence of better testing tools promoted earlier detection of problems and improved even detection of problems that would have otherwise gone 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 increasing early-backout rate was also due to the sheriff-managed integration repositories and their effect on how developers test their code. Before 2011, because developers pushed changes directly to m-c, the changes had to be thoroughly tested to avoid breaking the builds or introducing bugs. From 2011 to 2013, developers committed to integration repositories, and the sheriff backed out

problematic patches before merging changes to m-c, thus keeping it stable, as we mentioned before. So, developers were encouraged to commit to m-i after having performed less testing. As someone stated in Mozilla’s wiki,

But breaking it [the integration repository] rarely is ok. ... Never breaking the tree means you’re running too many tests before landing [committing to the repository].⁴

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.

So What?

To understand what the changes in backout rates mean to Firefox developers and users, we first have to understand the impact of early and late backouts.

The impact on developers. Every backout induces rework by requiring development of a new, improved patch. However, in Mozilla’s case, the increase of early backouts didn’t seem to cause overhead. Instead, it reflected a cultural change toward committing patches before testing them comprehensively, therefore reducing the effort required to test patches. Such change was possible only because broken patches no longer reached m-c. Sheriffs also ensured that patches that break the build were

backed out as soon as possible, reducing the time in which the repository must be closed:

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

Although all backouts induce rework, late backouts are severer. First, the longer a fix takes 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 aren't resolved early might end up in a release. So, users might have to wait another release cycle to receive the definitive bug fix. Finally, with integration repositories, inappropriate commits that weren't backed out early ended up in m-c, on which developers based their work. By the time the commit was backed out, many other commits might have depended on it.

So, from the shift toward earlier backouts, we can infer that the sheriff-managed integration branches reduced the effort required to integrate bug fixes.

The impact on users. Although 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 the then chair of Mozilla Foundation summarized on his blog post,

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 mak-

ABOUT THE AUTHORS



RODRIGO SOUZA is a PhD student in the Federal University of Bahia's Department of Computer Science. His research interests include empirical software engineering, release engineering, software evolution, and mining software repositories. Souza received an MSc in computer science from the Federal University of Campina Grande. Contact him at rodrigo@dcc.ufba.br.



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



ROBERTO A. BITTENCOURT is an assistant professor of computer engineering at the State University of Feira de Santana. His research interests include software evolution and design, computing education, and computer-supported cooperative work. Bittencourt received a PhD in computer science from the Federal University of Campina Grande. Contact him at roberto@uefs.br.

ing sure all aspects of the product and ecosystem were ready.⁵

However, backouts had no effect on users' perception of quality. This is because, after being committed to m-c, all bug fixes went through two other repositories, aurora and beta, where more tests occurred during two release cycles before they were released to the general public. So, only very late backouts affected users, and these were rare under both traditional and rapid releases. As a Mozilla engineer stated,

I think our development process gives us a margin of safety to detect regressions well before the code actually reaches 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.

As we mentioned before, Mozilla took two concrete measures that helped keep the process stable while letting it move faster: improving automated testing tools and using integration repositories. You can use these two measures to improve any project. However, the overhead incurred in implementing them is more justifiable under rapid releases because in that context it's important to keep the source code stable as often as possible. For example, having

FOCUS: RELEASE ENGINEERING

IEEE  computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

Next Board Meeting: 1-5 June 2015, Atlanta, GA, USA

EXECUTIVE COMMITTEE

President: Thomas M. Conte

President-Elect: Roger U. Fujii; **Past President:** Dejan S. Milojevic; **Secretary:** Cecilia Metra; **Treasurer, 2nd VP:** David S. Ebert; **1st VP, Member & Geographic Activities:** Elizabeth L. Burd; **VP, Publications:** Jean-Luc Gaudiot; **VP, Professional & Educational Activities:** Charlene (Chuck) Walrad; **VP, Standards Activities:** Don Wright; **VP, Technical & Conference Activities:** Phillip A. Laplante; **2015-2016 IEEE Director & Delegate Division VIII:** John W. Walz; **2014-2015 IEEE Director & Delegate Division V:** Susan K. (Kathy) Land; **2015 IEEE Director-Elect & Delegate Division V:** Harold Javid

BOARD OF GOVERNORS

Term Expiring 2015: Ann DeMarle, Cecilia Metra, Nita Patel, Diomidis Spinellis, Phillip A. Laplante, Jean-Luc Gaudiot, Stefano Zanero

Term Expiring 2016: David A. Bader, Pierre Bourque, Dennis J. Frailey, Jill I. Gostin, Atsuhiko Goto, Rob Reilly, Christina M. Schober

Term Expiring 2017: David Lomet, Ming C. Lin, Gregory T. Byrd, Alfredo Benso, Forrest Shull, Fabrizio Lombardi, Hausi A. Muller

EXECUTIVE STAFF

Executive Director: Angela R. Burgess; **Director, Governance & Associate Executive Director:** Anne Marie Kelly; **Director, Finance & Accounting:** John G. Miller; **Director, Information Technology Services:** Ray Kahn; **Director, Membership:** Eric Berkowitz; **Director, Products & Services:** Evan M. Butterfield; **Director, Sales & Marketing:** Chris Jensen

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928

Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614 • **Email:** hq.ofc@computer.org

Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720

Phone: +1 714 821 8380 • **Email:** help@computer.org

Membership & Publication Orders

Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • **Email:** help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-

0062, Japan • **Phone:** +81 3 3408 3118 • **Fax:** +81 3 3408 3553 • **Email:** tokyo.ofc@computer.org

IEEE BOARD OF DIRECTORS

President & CEO: Howard E. Michel; **President-Elect:** Barry L. Shoop; **Past**

President: J. Roberto de Marca; **Director & Secretary:** Parviz Famouri; **Director & Treasurer:** Jerry Hudgins; **Director & President, IEEE-USA:** James A. Jefferies;

Director & President, Standards Association: Bruce P. Kraemer; **Director & VP,**

Educational Activities: Saurabh Sinha; **Director & VP, Membership and Geographic**

Activities: Wai-Choong Wong; **Director & VP, Publication Services and Products:**

Sheila Hemami; **Director & VP, Technical Activities:** Vincenzo Piuri; **Director &**


Delegate Division V: Susan K. (Kathy) Land; **Director & Delegate Division VIII:**

John W. Walz

revised 27 Jan. 2015



frequently stable code is important when Mozilla must deliver a “chem-spill” release—one that fixes critical security issues and thus should reach users as soon as possible.

Our analysis uncovered previously unknown information about the evolution of early and late backouts in Firefox and therefore helped evaluate the impact 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. Release engineers would have easier access to up-to-date information about the process, letting them evaluate the impact of yet-to-be-made decisions. 

Acknowledgments

We thank all the Mozilla engineers who provided feedback on the results.

References

1. D. Zhang et al., “Software Analytics in Practice,” *IEEE Software*, vol. 30, no. 5, 2013, pp. 30–37.
2. “Releases,” *MozillaWiki*, Mozilla, 2014; <https://wiki.mozilla.org/Releases>.
3. “Committing Rules and Responsibilities,” Mozilla, 2014; https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Committing_Rules_and_Responsibilities.
4. “Tree Rules/Integration,” Mozilla, 2014; https://wiki.mozilla.org/Tree_Rules/Integration.
5. M. Baker, “Rapid Release Follow-Up,” blog, 3 Oct. 2011; <http://blog.lizardwrangler.com/?p=2996>.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.