On Rapid Releases and Software Testing

Mika V. Mäntylä[1], Foutse Khomh[2], Bram Adams[2], Emelie Engström[3], Kai Petersen[4]
[1] *Dep. of Computer Science and Engineering, Aalto University, Finland*
[2] *SWAT–MCIS, École Polytechnique de Montréal, Québec, Canada*
[3] *Dep. of Computer Science, Lund University, Sweden*
[4] *School of Computing, Blekinge Institute of Technology, Sweden*
mika.mantyla@aalto.fi, {foutse.khomh, bram.adams}@polymtl.ca, emelie.engström@cs.lth.se, kai.petersen@bth.se

***Abstract*—Large open and closed source organizations like Google, Facebook and Mozilla are migrating their products towards rapid releases. While this allows faster time-to-market and user feedback, it also implies less time for testing and bug fixing. Since initial research results indeed show that rapid releases fix proportionally less reported bugs than traditional releases, this paper investigates the changes in software testing effort after moving to rapid releases. We analyze the results of 312,502 execution runs of the 1,547 mostly manual system-level test cases of Mozilla Firefox from 2006 to 2012 (5 major traditional and 9 major rapid releases), and triangulated our findings with a Mozilla QA engineer. In rapid releases, testing has a narrower scope that enables deeper investigation of the features and regressions with the highest risk, while traditional releases run the whole test suite. Furthermore, rapid releases make it more difficult to build a large testing community, forcing Mozilla to increase contractor resources in order to sustain testing for rapid releases.**

***Keywords*-Software testing; release model; builds; bugs; open-source; agile releases; Mozilla.**

I. Introduction

Due to heavy competition, web-based organizations, both at the server side (e.g., Facebook and Google) and the client side (e.g., Google Chrome and Mozilla Firefox), have been forced to change their development processes towards rapid release models. Instead of working for months on a major new release, companies limit their cycle time (i.e., time between two subsequent releases) to a couple of weeks, days or (in some cases) hours to bring their latest features to customers faster [1]. For example, starting from version 5.0, Firefox has been releasing a new major version every 6 weeks [2].

Although rapid release cycles provide faster user feedback and are easier to plan (due to the smaller scope) [3], they also have important repercussions on software quality. For one, enterprises currently lack time to stabilize their platforms [4] and customer support costs are increasing because of the frequent upgrades [5]. More worrying are the conflicting findings that rapid release models (RRs) are either *slower* [6] or *faster* [7] at fixing bugs than traditional release models (TRs). Even in the latter case, the study still found that proportionally less bugs were being fixed, and that the bugs that were not fixed led to crashes earlier on during execution.

Since testing plays a major role in quality assurance, this paper investigates how RR models impact software testing effort. For example, Porter et al. noted that, since there is less time available, testers have less time to test all possible configurations of a released product, which can have a negative effect on software quality [8]. On the other hand, other studies have reported the positive effects of RRs on software testing and quality in the context of agile development, where testing has become more focused [9], [10]. To our knowledge, the impact of RRs on software testing measures, beyond defect data, have not yet been investigated.

Hence, to analyze the impact of RR models on the testing process, we analyze the system testing process in the Mozilla Firefox project, a popular web browser, and the changes it went through while moving from a TR model of one release a year to an RR model where new releases come every 6 weeks. We analyzed the system-level test case execution data from releases 2.0 to 13.0 (06/2006–06/2012), which includes five major TR versions (2.0, 3.0, 3.5, 3.6, and 4.0) with 147 smaller releases (20 alphas, 29 betas, 12 release candidates, and 86 minor), and nine RR versions (5.0 until 13.0) with 89 smaller releases (17 alphas, 56 betas, and 16 minor).

Based on this data and feedback from a Mozilla QA engineer, we studied the following four research questions:

RQ1) Do RRs affect the amount of testing performed?
RRs perform more test executions per day, but these tests focus on a smaller subset of the test case corpus.

RQ2) Do RRs affect the number of testers working on a project?
RRs have less testers, but they have a higher workload.

RQ3) Do RRs affect the frequency of testing activity?
RRs test fewer, but larger builds.

RQ4) Do RRs affect the number of configurations being tested?
RRs test fewer platforms in total, but test each supported platform more thoroughly.

A better understanding of the impact of the release cycle on testing effort will help software organizations to plan ahead and to safely migrate to an RR model, while enabling them to safeguard the quality of their software product.

The rest of the paper is organized as follows. Section II provides some background on Mozilla Firefox, while Section III describes the design of our study. Section IV presents the results of the study, followed by a discussion of these results

1063-6773/13 $26.00 © 2013 IEEE
DOI 10.1109/ICSM.2013.13

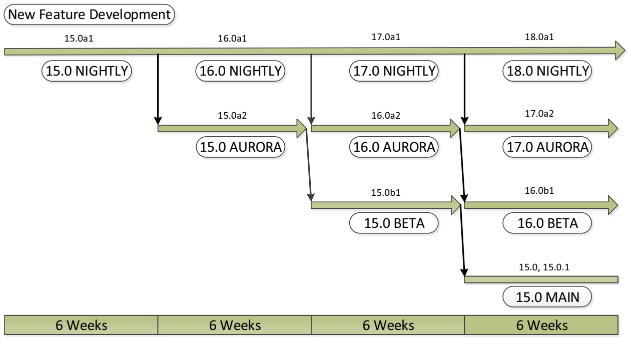


Figure 1: Rapid release process of Mozilla Firefox.

(Section V). Section VI relates our study with previous work. Finally, Section VII concludes the paper and outlines some avenues for future work.

II. Background

A. Firefox Development Process

Firefox is an open source web browser developed by the Mozilla Corporation. As of April 2013, Firefox has approximately 22% of web browser usage share worldwide [11], with almost half a billion users. Firefox 1.0 was released in November 2004 and the latest version, Firefox 21, was released on May 7, 2013, containing more than 8 MLOC (especially C++, C and JavaScript).

Firefox followed a traditional release model until version 4.0 (March 2011), after which it moved to a rapid release cycle from version 5.0 on, in order to compete with Google Chrome's rapid release model [4], [12], which was eroding Firefox's user base. Every TR version of Firefox was followed by a long series of minor versions, each containing bug fixes or minor updates over the previous version. However, in the RR model, every Firefox version now flows through four release channels (Figure 1): nightly, aurora (alpha), beta and main.

In RRs, the versions basically move from one channel to the next every 6 weeks [13]. The nightly channel integrates new features from the developers' source code repositories as soon as the features are ready. The aurora channel inherits new features from nightly at regular intervals (*i.e.*, every 6 weeks). The features that need more work are disabled and left for the next import cycle into aurora. The beta channel receives only new alpha features from aurora that are scheduled by management for the next Firefox release. Finally, mature beta features make it into main, i.e., the next official release.

B. Firefox Quality Assurance

Firefox heavily relies on contributors and end users to participate in the quality assurance (QA) effort. The estimated number of contributors and end users on the channels are respectively 100,000 for nightly, 1 million for alpha (aurora), 10 million for beta and 100+ millions for a major Firefox version [14]. Except for the automated Mozmill infrastructure for in-house regression testing, the testing done by the community is mostly manual.

To co-ordinate this community-based testing, Firefox has a system-level regression testing infrastructure called Litmus [15]. As explained by a Mozilla QA engineer, *"We use it primarily to test past regressions . . . and as an entry point for community involvement in release testing"*. It consists of a database with well-documented, functional test cases and stored execution results that are used to make sure that all functionality still works. Each test case corresponds to a user-visible feature, for example "Standard installation", "Back and Forward buttons", and "Open a new window". The test case for "Back and Forward buttons" states: "Steps to perform: 1) Visit two successive sites. 2) Click Back button twice. 3) Click Forward button twice. Expected Results: page loading should move back and forward through history as expected".

The interface of Litmus is a web-based GUI that allows contributors to follow the status of currently running or archived test executions, and to submit the results of manual tests. Furthermore, users can consult the error messages generated during failing test runs. Litmus is used mainly for beta, release candidate, main, and minor versions, but much less frequently for alpha releases (only 27% of them used Litmus). The pass percentage for test executions in Litmus is around 98%.

III. Study Design

In order to address the four research questions presented in the introduction, we mined the test execution data stored in Firefox' Litmus repository, then performed various analyses on this data. The remainder of this section elaborates on our data collection and analysis.

A. Data Collection

We performed web crawling of the Litmus system to get the test cases and the execution data of the test cases for Firefox versions 2.0 to 13.0. Overall, we identified 1,547 unique test cases (roughly 10% of them are automated) for a total of 312,502 test case executions across 6 years of testing (06/2006–06/2012), performed by 6,058 individuals on 2,009 software builds, 22 operating system versions and 78 locales. During this time frame, the Firefox project made 249 releases, of which 213 releases (142 TR and 71 RR) reported their testing activity into the Litmus system. We only consider data until June 2012, since immediately afterwards Litmus was replaced by the Moztrap system in order to enable adding new test cases in a collaborative way and scaling up in terms of usability and functionality [16]. The transition to Moztrap happened instantaneously from one version to the next, which means that our data is not biased by this transition.

In Litmus, all test cases provide the following information: major version number of the release (2.0–13.0), unique identifier, summary, regression bug identifier (if any), the test steps to perform, the expected results, the test group and subgroup the test case belongs to (if any), and links to the corresponding test execution results. Each test execution contains the following information: status ("pass"/"fail"/"test

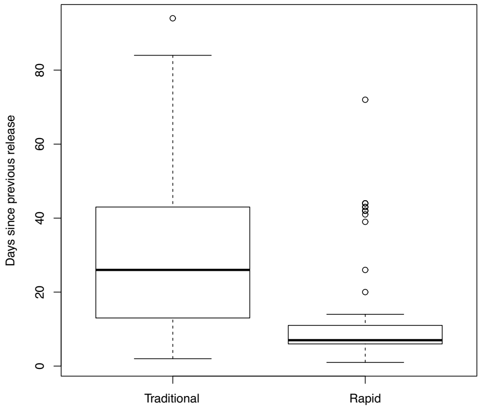


Figure 2: Distribution of release cycle length (in days) for TRs and RRs.

unclear or broken"), test case identifier, time-stamp, platform (e.g., Windows), operating system (e.g., Windows XP), build identifier, locale, user agent, referenced bugs (if any), comments (if any) and the test logs (if any). However, there was no explicit information of the exact release (alpha, beta, release-candidate, major or minor) each test execution was related to. Hence, we mapped the test execution time-stamps to the release dates, which were available online [17].

For each of the analyzed versions, we also extracted the code revision history from the Mercurial repository [18] and parsed it to extract information about the frequency and number of commits. Finally, to triangulate our findings we performed an email interview with a Mozilla QA engineer who has been working on QA for the Firefox project for the past five years. Although the interview results are based on the views of one Mozilla employee (and hence might be incomplete or contain small inaccuracies), they were consistent with our analysis results and provide insights into some of our empirical findings.

B. Data Analysis

We analyze the collected data set using a set of metrics defined specifically for each question. Details of the metrics are provided later on in the sections discussing the research questions. We use the R statistical analysis tool to perform all the calculations. The Shapiro-Wilk test showed that our data is not normally distributed. Therefore, we use non-parametric statistical analysis throughout the paper. To compare between two groups, we use the Wilcoxon rank-sum test (WRST) and to study effect sizes we use Cliff's delta (provided by the R package orrdom [19]), which varies between -1 and 1.

As the length of software release projects can vary greatly, we normalized all metrics for project duration (measured in days) to make statistical comparison between TR and RR releases possible. In the RR model, the median time between releases is 7 days while in the TR model it is 26 days, if we consider all types of releases (i.e., alpha, beta, release candidate, major and minor) together. This difference is statistically significant with a large effect size, see Table I. Note that this normalization does not apply to Figure 4, Figure 5, Figure 7 and Figure 9, which show cumulative growth.

Table I: Comparison between metrics for the TR and RR. Effect size uses Cliff's Delta.

	TR	RR	WRST	Effect
	(median)	(median)	(p-value)	size
Release length	26.00	7.00	**0.000**	-0.586
Test exec. per day (RQ1)	50.86	127.3	**0.000**	0.359
Testcases per day (RQ1)	10.65	14.00	0.855	0.015
Testers per day (RQ2)	1.667	1.000	**0.000**	-0.297
Builds per day (RQ3)	0.427	0.250	**0.004**	-0.242
Locales per day (RQ4)	0.302	0.143	**0.000**	-0.356
OSs per day (RQ4)	0.270	1.286	**0.000**	0.695

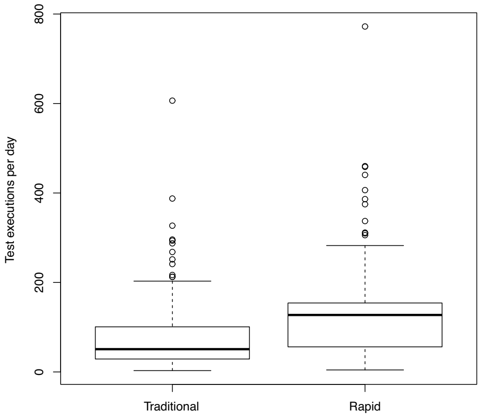


Figure 3: Distribution of number of test executions per day for TRs and RRs.

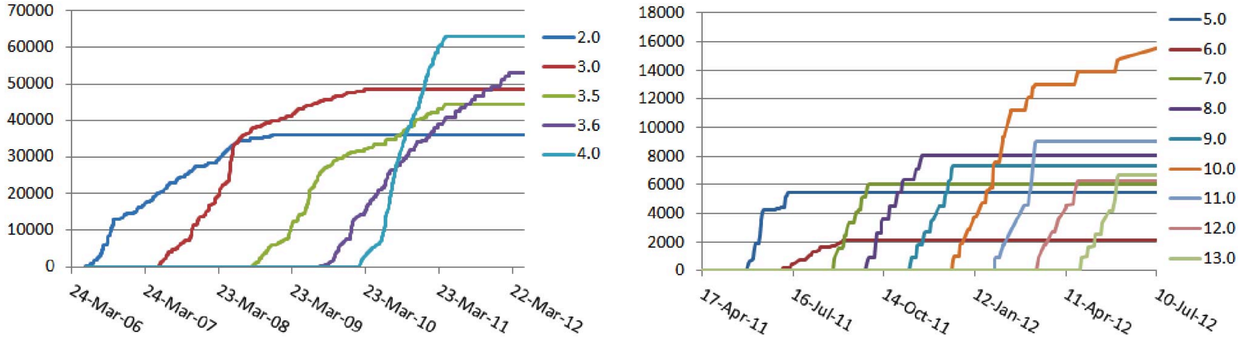
IV. Case Study Results

This section discusses for each research question, its motivation, the approach we used, our findings and the feedback by the interviewed QA engineer.

RQ1) Do RRs affect the amount of testing performed?

Motivation: In our previous work [7], we found that RR models fix bugs faster than TR models, but fix proportionally less bugs. Since the short release cycle time of RR models seems to allow less time for developers to test the system, QA teams could decide to reduce the amount of testing for their RR versions in order to cope with their tight schedule. In this research question, we verify this by investigating the amount of testing effort performed for each TR and RR version of Firefox.

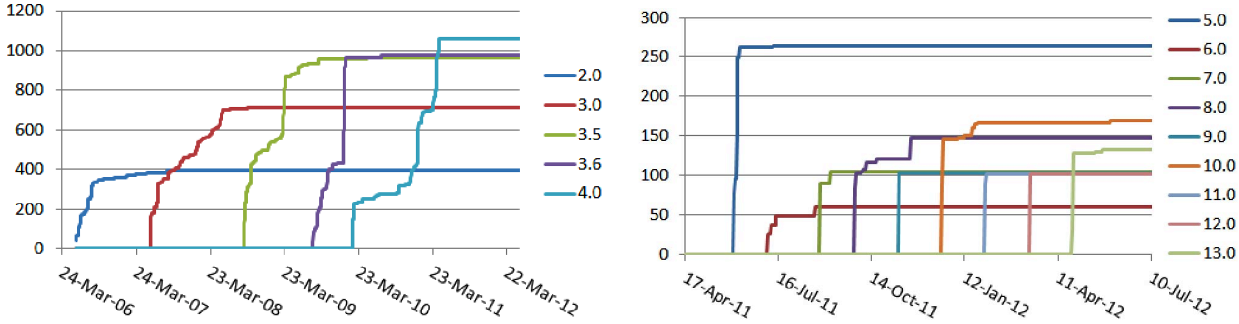
Null Hypotheses: We test the following two null hypotheses to compare the amount and the functional coverage of tests executed for TR and RR models:
H^1_{01}: There is no significant difference between the number of tests executed per day for RR releases and TR releases.
H^1_{02}: There is no significant difference between the number of unique test cases executed per day for RR releases and TR releases.



(a)

(b)

Figure 4: Cumulative number of test executions over time (not normalized) for (a) TR and (b) RR releases.



(a)

(b)

Figure 5: Cumulative number of unique test cases executed over time (not normalized) for (a) TR and (b) RR releases.

We use the Wilcoxon rank-sum test [20] to test H_{01}^1 and H_{02}^1 using a confidence level of 1% (*i.e.*, p-value < 0.01).

Metrics: For each alpha, beta, release-candidate, major and minor version of Firefox in our data set, we compute the following two metrics that capture the amount of tests executed and the functional coverage of the tests. Functional coverage is the degree to which the different features in a software version are tested by a test suite. The more unique (i.e., functionally different) test cases are being executed, the higher the functional coverage.

- #Test exec. per day: the number of tests executed per day.
- #Test cases per day: the number of unique test cases executed per day.

Findings: **The RR model executes almost twice as many tests per day (median) compared to TR models.** Figure 3 and Table I show a statistically significant difference between the two distributions, with a medium effect size of 0.359 (Cliff's delta), i.e., RR models run more tests in a shorter time frame. Therefore, we reject H_{01}^1. This difference is also clear from Figure 4, which shows the cumulative (absolute) number of test executions for each major TR and RR releases over time. Since alpha releases typically are not tested in Litmus, the data for each RR starts from the first beta release (this holds for all cumulative plots in this paper). The number of test executions obtains a much higher absolute value (between 35,000 and 50,000, except for the 4.x release series) than the RR releases (usually smaller than 9,000, except for release 10.0), but accumulates over a much longer time.

Firefox 10.0 is an exception for RRs, since it is the first "Extended Support Release" (ESR) [21], i.e., it is meant to last for 54 weeks instead of 6 (lifetime of 9 "normal" RRs). An ESR helps corporate clients [22] to certify and standardize on one particular browser version for a longer period, while still receiving security updates (backported from more recent non-ESR releases). We can see that testing for 10.0 evolved linearly until its release, after which testing is resumed only shortly before the release of a new version. In between, no testing occurs for 10.0.

Especially for the TR releases, testing continues even though development on a newer version has already started. This is due to the many minor releases that follow a major TR release. For RR releases, testing of the next release starts soon after the testing for the previous release stops. Release 10.0 again is an exception, since testing needs to continue for 9 releases. All releases (TR and RR) see accelerated test execution right before a release, which is visible as an almost vertical trend in Figure 4.

Similar functional test coverage per day, but lower coverage overall. Data exploration revealed that the test cases executed for each major release vary based on the features implemented in the release. The median similarity of functional test coverage between subsequent major releases was 56% for both TRs and RRs. We counted the number of unique test cases executed for a particular release, then divided this by the length of the release cycle to compare the number of unique test cases per day executed by each release. We found that, on average, the number of unique test cases executed per day is slightly higher in RR. However, this difference is not statistically significant and the effect size is almost non-existent (0.015). Therefore, we cannot reject H^1_{02}. This means that, although a particular test case gets more executions in absolute numbers for TR releases, it will be executed more frequently in a shorter time frame for RR releases.

However, since there is less time to run tests, RR testers limit the scope (and hence coverage) of their tests to only the most important ones. Figure 5 shows for each TR and RR release the evolution over time of the cumulative number of unique test cases being executed. The number of different tests executed increases monotonically across the major TR releases, i.e., Firefox 4.x was tested on more cases (almost 1,100) than Firefox 3.5.x and earlier releases. However, the RR releases seem to be tested on progressively less different test cases, going from 270 unique test cases for Firefox 5.0 down to 100 for Firefox 12. Most RR releases reach 90% of their maximum number of unique test cases within a week, which indicates that the reduced scope of testing is determined very early during a new release cycle.

Feedback QA engineer The QA engineer could not confirm the difference in the number of test executions, but strongly supported our finding that testing is more focused: *"To survive under the time contraints of a rapid release we've had to cut the fat and focus on those test areas which are prone to failure, less on ensuring legacy support"*. In particular, the focused test set consists of *"a fixed set of tests for areas prone to regression (Flash plugin testing for example)"* and *"a dynamically changing set of tests to cover recent regressions we chemspilled for and high risk features"*. A "chemspill" is a negative event like a vulnerability that requires a quick update. Overall, the QA engineer believed that the narrow scope of RR tests is highly beneficial: *"The greatest strength is that the scope of what needs to be tested is narrow so we can focus all of our energy on deep diving into a few areas"*.

The amount of test executions per day is significantly larger in RR, but these tests focus on a smaller subset of the test case corpus instead of on the full corpus.

RQ2) Do RRs affect the number of testers working on a project?

Motivation: With short release cycles, development teams have less time to implement new features and test the features before they are released to users. In **RQ1**, we observed on the one hand a reduction in functional coverage, while on the other hand the remaining test cases are executed more frequently in the shorter time between two releases. Given these observations, does the same testing team as before handle testing, with each tester having to perform less work, or did the test team shrink, either because there is less work to do, or because the rapid succession of releases makes it harder to retain testers?

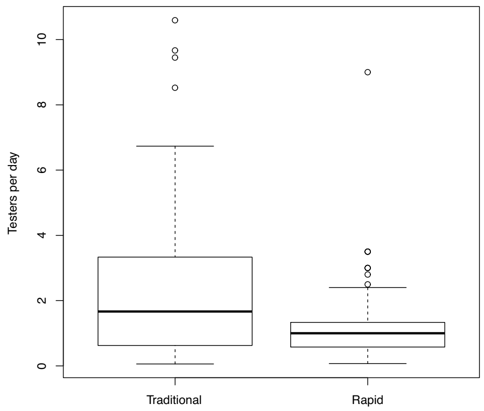


Figure 6: Distribution of number of testers per day for TRs and RRs.

Null Hypotheses: We test the following null hypothesis to compare the number of testers for TR and RR releases:
H^2_{01}: There is no significant difference between the number of testers for RR releases and TR releases.
Similar to **RQ1**, we use the Wilcoxon rank-sum test [20] to test H^2_{01} using a 1% confidence level.

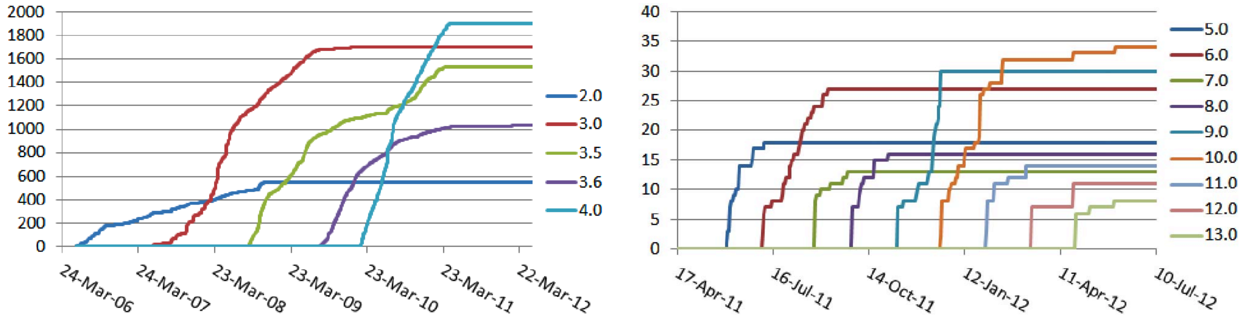
Metrics: For each alpha, beta, release-candidate, major and minor version of Firefox in our data set, we compute the following metric:

- #Testers per day: the number of testers per day.

Findings: **Fewer testers conduct testing for RR releases.** Figure 6 shows the distribution of the number of individuals per day testing the traditional and rapid releases. We can see that TR releases have a median of 1.67 testers per day compared to 1.0 testers per day for RR releases. The Wilcoxon rank-sum test yields a statistically significant result, i.e., we can reject H^2_{01}. This result in conjunction with the results of the previous section means that the average workload per individual is much higher under an RR model, since more tests need to be executed per day by less people (*i.e.*, median of 35 vs. 120 test executions per tester per day).

Figure 7a and Figure 7b by themselves do not show a clear trend, with some releases having significantly more testers than others. However, the contrast between TR and RR releases again is very stark when measured from the Litmus system. The most heavily tested RR releases (ESR release 10.0) reached 34 testers by July 2012, which is a factor 56 lower than the 1,900 different testers for version 4.x. Overall, TR releases had a total of 6,010 unique testers, while for RR releases there were only 105 unique software testers registered in the Litmus system.

One possible hypothesis is that the drop in number of testers can be explained by an increase in test automation. For example, across the analyzed Firefox history, we found



(a)

(b)

Figure 7: Cumulative number of unique testers running Litmus tests for (a) TR and (b) RR releases.

that 158 out of the 1,547 test cases have been executed by the "#mozmill" username (corresponding to the name of the automated regression testing system). However, 16 of those test cases have been executed only once by "#mozmill", suggesting a failed automation attempt. Furthermore, all 158 test cases had also been executed with other usernames, suggesting that sometimes the test is run manually (the automated tests contained detailed instruction for manual execution), perhaps due to the test breaking down. However, if changes in the share of test automation would have dramatically impacted testing, this should have led to a significant positive correlation between the evolution of the project and the number of test executions (or cases) per day. Section V-A shows that this is not the case.

Feedback QA engineer The interview confirmed our statistical findings about the decreasing number of testers: *"The weakest point [in RR] is that it's harder to develop a large community which more accurately represents the scale and variability of the general population. Frequently this means that we don't hear about issues until after release, in effect turning our release early adopters into beta testers"*. To counter this, Mozilla has augmented their core testing team with contractors: *"1) the core team has remained largely unchanged since adopting rapid release 2) the contract team has nearly doubled . . . We can scale up our team much faster through contractors than through hiring. The time afforded to us to make the switch to rapid releases left little room for failure which is why we took that approach."*. The number of testers has also been impacted by some competing Mozilla projects. Regarding test automation, the QA engineer noted that *"many of our Litmus tests have partial coverage across our various automation frameworks"*, but that after the switch to Moztrap all automated tests were left out. Furthermore, he confirmed that *"I think it's impossible to say how much [test automation] coverage we have for sure [in Litmus]"*.

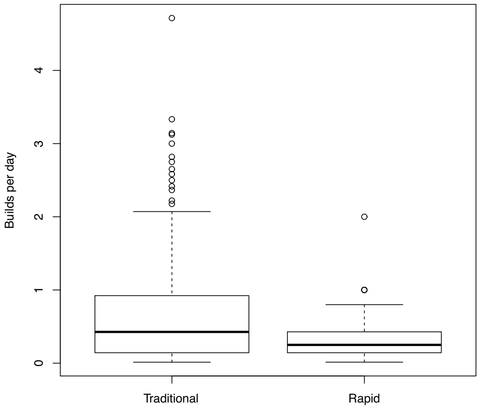


Figure 8: Distribution of the number of builds tested per day for TRs and RRs.

The migration to the RR model has reduced the community participation in testing when adjusting for project duration. However, to keep up with the rapid releases the number of specialized testing resources has increased.

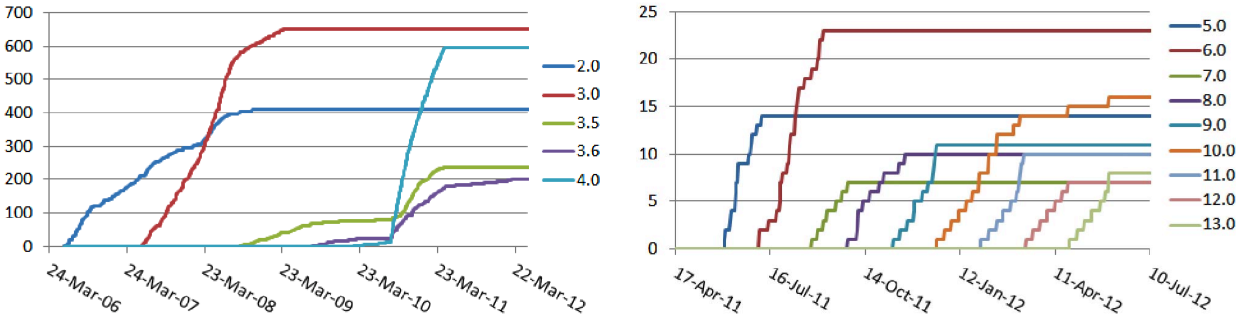
RQ3) Do RRs affect the frequency of testing activity?

Motivation: Given that more tests are executed per day for RR releases, this could be explained because comparatively more intermediate builds that need testing are produced in a shorter time frame. In other words, developer productivity could have increased compared to TR releases, requiring more tests to be run. Alternatively, maybe the number of builds did not increase significantly, but the amount of change between builds has increased, requiring more testing to be performed on each build. This research question investigates these hypotheses.

Null Hypotheses: We test the following null hypotheses:

H^3_{01}: There is no significant difference between the number of tested builds per day for RR releases and TR releases.

H^3_{02}: There is no significant difference between the number of commits per day for RR releases and TR releases.



(a) (b)

Figure 9: Cumulative number of unique builds for which tests have been run for (a) TR and (b) RR releases.

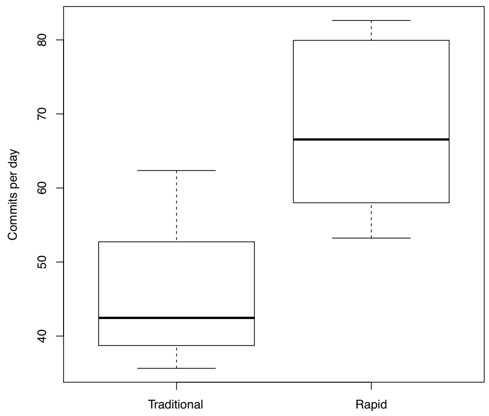


Figure 10: Distribution of the number of commits per day for TRs and RRs.

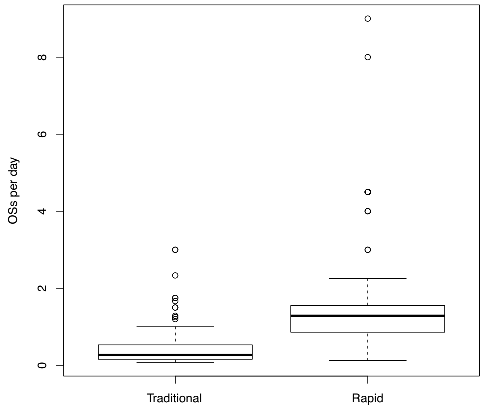


Figure 11: Distribution of number of operating systems tested per day for TRs and RRs.

We again use the Wilcoxon rank-sum test [20] to test these null hypotheses using a 1% confidence level.

Metrics: We calculated the following metrics:

- #Tested builds per day: the number of tested builds per day
- #Commits per day: the number of commits to the Mercurial repository per day

We calculated the #Tested builds per day for each alpha, beta, release-candidate, major and minor version of Firefox in our data set, while we calculated the #Commits per day only for the major release, since commits are hard to link to specific releases.

Findings: **Less rapid release builds are being tested per day.** Figure 8 shows that the number of tested RR builds per day is statistically significantly lower than the number of tested TR builds per day (0.25 vs. 0.427). We reject H_{01}^{3}. In other words, the higher relative frequency of test executions cannot be explained by more builds being tested, but it seems that each build is tested more thoroughly (albeit with a smaller coverage, see **RQ2**).

RR builds contain more code commits than TR builds. To understand why less RR builds are being tested, we analyzed whether these builds contain more commits relative to TR builds. Figure 10 compares the distribution of the number of commits per day for all major TR and RR releases. RR releases result in statistically significantly more commits per day than the TR releases. Hence, we can reject H_{02}^{3}. Since our data exploration showed a downward trend across time in the number of commits, more commits are integrated into version control in a shorter time frame.

Feedback QA engineer The QA engineer had not noticed any difference between the number of builds tested between TR and RR releases. However, he agreed that RRs contained more changes, but he attributed this observation more to the project's evolution than to the RR model: *"As time has gone on we have increased the number of changes that land per day"*.

RR releases focus testing on fewer, but larger builds when adjusted for release duration.

RQ4) Do RRs affect the number of configurations being tested?

Motivation: The final dimension that we study are the different configurations that are tested, such as different operating systems or support for more locales (i.e., language settings and internationalization [23]). More thorough testing of different configurations could explain the larger number of tests per build (**RQ3**), as well as the higher workload of individual testers (**RQ2**).

Null Hypotheses: We test the following null hypotheses:
H^4_{01}: There is no significant difference between the number of tested locales per day for RR releases and TR releases.
H^4_{02}: There is no significant difference between the number of tested operating systems per day for RR releases and TR releases.
We again use the Wilcoxon rank-sum test [20] to test these null hypotheses using a 1% confidence level.

Metrics: We calculated the following metrics:

- #Tested locales per day: the number of tested locales per day
- #Tested operating systems per day: the number of operating systems tested per day

We calculated these metrics for all alpha, beta, release-candidate, major and minor releases of Firefox in our data set.

Findings: **RR tests are conducted on only one locale manually.** When comparing the distribution of the number of tested locales per day, we found that the number of RR locales tested is only half the number of TR locales tested (0.302 vs. 0.143). It should be noted that the locale “English US” dominates the number of test executions in all TR and RR releases. The average share of test executions of the English US locale for TR models is 91%, compared to 99% for RR models.

A slightly lower number of platforms is being tested, but more thoroughly. Figure 11 shows that the number of operating systems tested per day has increased by almost 400% (median of 0.27 vs. 1.286) when moving to RR releases. However, the total number of tested operating systems has dropped slightly, with most of the RR releases testing 9 operating systems compared to 12 to 17 for TR releases. This can be partly attributed to the longer time frame of the TRs, e.g., if a major release is tested over a two years period versus 6 weeks (see Figure 4) it is far more likely that new operating system versions enter the market during the longer time period.

Furthermore, when looking at the detailed execution data per operating system, we found for each RR release that all tested operating systems get roughly the same amount of test executions. For TR releases, there were large fluctuations in the number of executions between the tested operating systems.

Feedback QA engineer The interview revealed that locale test coverage has actually increased, but has been entirely converted to automated tests, disappearing out of the scope of the Litmus system. Furthermore, the total number of operating systems tested has decreased because *“we now distribute across Betas. For example, we might test Windows 7, OSX 10.8, and Ubuntu in one Beta then Windows XP, Mac OSX 10.7, and Ubuntu in another Beta”.*

Table II: Kendalls’s tau correlation between release model, length and date. Significance levels *=0.05 **=0.01, ***=0.001.

	Release length	Project Evolution
Release model	-0.397***	0.634***
Release length	N/A	-0.294***

Table III: Partial correlation of three variables (release model, length and date) to test effort, while controlling two out of the three variables. Significance levels *=0.05 **=0.01, ***=0.001.

	Partial Kendall correlation coefficients		
RQ	Release model	Release length	Project Evolution
#Test executions per day (RQ1)	0.026	-0.414***	0.048
#Test cases per day (RQ1)	-0.231***	-0.593***	0.017
#Testers per day (RQ2)	-0.224***	-0.331***	-0.069
#Builds per day (RQ3)	-0.105*	-0.258***	-0.176***
#Locales per day (RQ4)	-0.281***	-0.443***	-0.115*
#OSs per day (RQ4)	0.149**	-0.822***	0.130**

RR releases test less locales manually. Each supported operating system is tested more thoroughly, but spread across beta releases.

V. Discussion

A. Confounding factors

During our empirical study, we realized that there are two important confounding factors that may affect the results: release length and the project’s natural evolution. First, one could easily think that the differences observed between TR and RR are not due to the release model, but due to the release cycle length, since even some of the TR releases have a shorter release cycle (see Figure 2). Second, the evolution of the project refers to the natural changes and events occurring over time, which are not necessarily related to release length or release model. For example, the reduction in the number of testers over time could be due to re-organization across competing projects or to a loss in community interest. To complicate matters more, both of these confounding factors are impacted significantly by the choice of release model, as the correlations between these variables show (Table II). In these calculations, release length (days between releases) and project evolution (the time-stamp of each release date) are simply modelled as continous numeric variables, while release model is nominal, set either to zero for TRs or one for RRs.

Hence, we investigated the effect of release model, release length, and project evolution on the metrics calculated for the four research questions, while controlling the confounding effect that these variables have on each other. For this, we used partial correlation (R package ppcor [24]), in which the correlation of one of the RQs’ metrics between RRs and TRs is measured, while controlling two variables out of release model, release length, and project evolution. We used the non-parametric Kendall’s tau instead of linear multiple regression, since the data is not normally distributed.

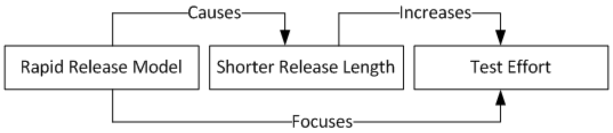


Figure 12: Model explaining the relationship between release model, release length and test effort.

Table III shows that the release model has a significant effect for five out of the six metrics when controlling for the release length and project evolution. It appears that the number of testers, test cases, builds, and locales tested per day are significantly smaller in the RR releases, while the number of operating systems per day significantly increases in the RR releases. On the other hand, the larger number of test executions per day for RR releases is not statistically significant when controlling for release length and project evolution. Instead, the effect that we observed in **RQ1** seems to be due to the consistently shorter time in between releases.

Regarding release length, the results show that test effort overall increases when release length shrinks, i.e., testing becomes more work. One hypothesis, supported by the feedback that we received, is that for the short TR releases a fixed set of regression tests needs to be run, regardless of the release duration. Another hypothesis is that the shorter TR releases are more often rapid patches used to quickly fix major bugs. In such a scenario, the changes in code are small, but as the release is going for millions of users they must be thoroughly regression-tested.

Finally, for project evolution, we find that the number of different locales and builds tested has decreased over time, while the number of operating systems tested has increased over time. No statistically significant change in number of testers can be observed.

Taking all of this into account, our interpretation of the relation between release model and test effort is depicted in Figure 12. Our initial hypothesis was that the shorter release length of RR releases was responsible for increased testing effort. However, even after controlling for the effect of the RRs' shorter release length, the RRs still have a lower number of test cases, testers, tested builds and tested locales. This means that Firefox' development process must have changed, as supported by the received feedback, since otherwise one would actually expect a higher proportion of the above metrics. The change in process has focused the testing efforts for the RR releases compared to the TR releases. Only the increase in test executions per day can be fully attributed to the shorter release length.

B. Limitations

Every empirical study has limitations. First, we cannot be certain that the changes that we see in the TR and RR metrics are caused by the change from TRs to RRs, which affects the construct validity of this study. After all, there could always be hidden factors that actually cause these observed differences, such as the competing projects in RQ2. To control for this, we triangulated our findings with a Mozilla engineer, who confirmed most of our findings.

Second, although we study over 200 Firefox releases, our study only considers one open source system, which affects the external validity of the study. However, the test data used for this study is not straightforward to obtain, even for open source systems, while for closed source systems substantial data is sealed within corporate walls. Nevertheless, more studies are needed before affirmative conclusions on the effects of a release model on testing effort can be made.

Third, the Litmus database only represents a part of the Firefox testing process, which is mostly aimed at manual regression testing of risky regression test cases and as entry point for community members to join testing. Since we did not study the automated regression test infrastructure, we cannot provide a complete picture of the Firefox testing process. This affects the internal validity of this study.

VI. Related Work

To the best of our knowledge, this study is the first attempt to empirically quantify the impact of release cycle time on testing in a real world setting with a large quantitative data set. This section looks at the literature on migration from TR to RR release models, followed by work on the impact of agile processes on testing.

A. Rapid releases and software quality

In recent years, many modern commercial software projects [25], [26] and open source projects backed by a company [12], [27] have switched towards shorter release cycles. Tool builders and researchers (e.g., [8]) have focused especially on enabling continuous delivery [28]. Amazon, for example, deploys on average every 11.6 seconds [26], achieving more than 1,000 deployments per hour.

However, the impact of rapid releases on the quality of the software product experienced by the end user has not been studied until recently. Baysal et al. [6] compared the release and bug fix strategies of Mozilla Firefox 3.x (TR) and Google Chrome (RR) based on browser usage data from web logs. Although the different profiles of both systems make direct comparison hard, the median time to fix a bug in the TR system (Firefox) seemed to be 16 days faster than in the RR system (Chrome), but this difference was not significant.

Khomh et al. [7] studied the impact of Firefox' transition to shorter development cycles on software quality and found no significant difference in the number of post-release defects, except that proportionally less defects were fixed (normalizing for the shorter time between releases). This suggests that Mozilla's strategies for testing RR, such as more contractor testing, more focused testing and alternating beta-release testing for operating systems, have been succesful in assuring the quality. However, the larger testing community in the TR era might have been able to find some errors earlier than is the case now, which could explain Khomh et al.'s findings about faster crashes in RR versions.

B. Process changes

Petersen and Wohlin [9] showed that early and continuous testing has a positive effect on fault-slip-through when migrating from plan-driven to agile development with faster releases. They reported on the improvements of test coverage at unit-test level, but highlighted that test cycles are often too short to conduct an extensive system test of quality attributes (e.g., performance), as these are more time intensive. We found that RR reduced the scope of testing, but allowed deeper testing within that scope. Earlier research [29] found that frequent deliveries to subsystem testing allowed earlier and more frequent feedback on each release, and increased the developers' incentives to deliver higher quality.

Li et al. [10] investigated the effect on product quality of introducing Scrum with a 30-day release cycle. They found that the quality focus had improved due to regular feedback for every sprint, better transparency, and an improved overview of remaining defects, leading to a timely (i.e., improved) removal of defects. Kettunen et al. [30] studied the differences in testing activities due to differences in process models. They found that early testing leads to more test execution time and a need for more predictable test resources. Our results support the finding that RR leads to more predictable test resource allocation.

VII. Conclusion

This paper has presented a case study on the effects of moving from traditional to rapid releases on Firefox' system testing. By triangulating data from the Litmus regression testing database with feedback from an interview with a Mozilla QA engineer, we make four key findings. First, we found that due to time-contraints RR system tests have a smaller scope and that the RR model has forced the Firefox testing team *"to cut the fat and focus on those test areas which are prone to failure"*. This narrow scope allows deeper testing in selected areas, which was seen as one of the largest strengths of RR testing. Second, we found that the number of specialized testers has grown due to an increase in the number of contractors, which were needed to sustain testing effort in the rapid release model. However, at the same time the large testing community which *"represent the scale and variability of the general population"* has decreased. Third, comparison to [7] shows that these rather significant changes in testing process have not significantly impacted the product quality. Fourth, based on empirical data we have proposed a theoretical model explaining the relationship between release model, release length and test effort that needs to be validated in future case studies.

Acknowledgment

We would like to thank the Mozilla QA engineer who provided feedback to our findings. The responses by Mozilla employees in this paper are accounts of personal experience and opinion, and are in no means whatsoever an official statement from Mozilla.

References

[1] HP, "Shorten release cycles by bringing developers to application lifecycle management," *HP Applications Handbook, Retrieved on Febuary 08, 2012*, 2012. [Online]. Available: http://bit.ly/x5PdXl

[2] InvestmentWatch, "Mozilla puts out firefox 5.0 web browser which carries over 1,000 improvements in just about 3 months of development," http://bit.ly/aecRrL, June 2011.

[3] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley, 2004.

[4] S. Shankland, "Rapid-release firefox meets corporate backlash," http://cnet.co/ktBsUU, June 2011.

[5] M. Kaply, "Why do companies stay on old technology?" *Retrieved on January 12, 2012*, 2012. [Online]. Available: http://bit.ly/k3fruK

[6] O. Baysal, I. Davis, and M. W. Godfrey, "A tale of two browsers," in *Proc. of the 8th Working Conf. on Mining Software Repositories (MSR)*, 2011, pp. 238–241.

[7] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *MSR*, 2012, pp. 179–188.

[8] A. Porter, C. Yilmaz, A. M. Memon, A. S. Krishna, D. C. Schmidt, and A. Gokhale, "Techniques and processes for improving the quality and performance of open-source software," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 163–176, 2006.

[9] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices," *Empirical Softw. Engg.*, vol. 15, no. 6, pp. 654–693, Dec. 2010.

[10] J. Li, N. B. Moe, and T. Dybå, "Transition from a plan-driven process to scrum: a longitudinal case study on software quality," in *Proc. of the 2010 ACM-IEEE Intl. Symp. on Empirical Software Engineering and Measurement (ESEM)*, 2010, pp. 13:1–13:10.

[11] R. S. Ltd., "Web browsers (global marketshare)," http://bit.ly/81klgi, April 2013.

[12] S. Shankland, "Google ethos speeds up chrome release cycle," http://cnet.co/wlS24U, July 2010.

[13] D. Sicore, "New channels for firefox rapid releases," http://bit.ly/hc1zmY, April 2011.

[14] R. Paul, "Mozilla outlines 16-week firefox development cycle," http://bit.ly/fLHEfo, March 2011.

[15] Mozilla, "Litmus wiki," http://mzl.la/evJmTW, January 2013.

[16] ——, "Moztrap wiki," http://bit.ly/XBGMfu, January 2013.

[17] Wikipedia, "Firefox release history," http://bit.ly/Ngvfln, January 2013.

[18] Mozilla, "Mozilla source code mercurial repositories," 2013. [Online]. Available: http://hg.mozilla.org/

[19] J. J. Rogmann, "orddom: Ordinal dominance statistics," http://bit.ly/Y0K0eo, January 2013.

[20] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*, 2nd ed. John Wiley and Sons, inc., 1999.

[21] Wikipedia, "Extended support release," http://bit.ly/ZlgqoM#Extended_Support_Release, January 2013.

[22] R. Paul, "Firefox extended support will mitigate rapid release challenges," http://ars.to/M2TbFQ, January 2012.

[23] Wikipedia, "Locale," http://bit.ly/2iJLwB, January 2013.

[24] S. Kim, "ppcor: Partial and semi-partial (part) correlation," http://bit.ly/XkWuyn, October 2012.

[25] A. W. Brown, "A case study in agile-at-scale delivery," in *Proc. of the 12th Intl. Conf. on Agile Processes in Software Engineering and Extreme Programming (XP)*, vol. 77, May 2011, pp. 266–281.

[26] J. Jenkins, "Velocity culture (the unmet challenge in ops)," Presentation at O'Reilly Velocity Conference, June 2011.

[27] E. Gamma, "Agile, open source, distributed, and on-time – inside the eclipse development process," Keynote at the 27th Intl. Conf. on Software Engineering (ICSE), May 2005.

[28] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[29] K. Petersen and C. Wohlin, "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case," *J. Syst. Softw.*, vol. 82, no. 9, pp. 1479–1490, Sep. 2009.

[30] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, "A study on agility and testing processes in software organizations," in *Proc. of the 19th Intl. Symp. on Software Testing and Analysis (ISSTA)*, 2010, pp. 231–240.