Empirical Software Engineering journal manuscript No.
(will be inserted by the editor)

On Rapid Releases and Software Testing: A Case Study and a Semi-Systematic Literature Review

Mika V. Mäntylä, Bram Adams, Foutse Khomh, Emelie Engström, Kai Petersen

Received: date / Accepted: date

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 in the context of a case study on Mozilla Firefox, and performs a semi-systematic literature review. The case study analyzes 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 triangulates our findings with a Mozilla QA engineer. We find that rapid releases have a narrower test scope that enables a deeper investigation of the features and regressions with the highest risk. Furthermore, rapid releases make testing more continuous and have proportionally smaller spikes before the main release. However, rapid releases make it more dif-

Mika V. Mäntylä
Dep. of Computer Science and Engineering,
Aalto University, Finland
E-mail: mika.mantyla@aalto.fi

Bram Adams
MCIS, Polytechnique Montréal,
Québec, Canada
E-mail: bram.adams@polymtl.ca

Foutse Khomh
SWAT, Polytechnique Montréal,
Québec, Canada
E-mail: foutse.khomh@polymtl.ca

Emelie Engström
Dep. of Computer Science,
Lund University, Sweden
E-mail: emelie.engstrom@cs.lth.se

Kai Petersen
School of Computing,
Blekinge Institute of Technology, Sweden
E-mail: kai.petersen@bth.se

ficult to build a large testing community, and they decrease test suite diversity and make testing more deadline oriented. In addition, our semi-systematic literature review presents the benefits, problems and enablers of rapid releases from 24 papers found using systematic search queries and a similar amount of papers found through other means. The literature review shows that rapid releases are a prevalent industrial practice that are utilized even in some highly critical domains of software engineering, and that rapid releases originated from several software development methodologies such as agile, open source, lean and internet-speed software development. However, empirical studies proving evidence of the claimed advantages and disadvantages of rapid releases are scarce.

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

1 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 their 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 the best of 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 Firefox QA engineer, we studied six research questions with respect to Mozilla Firefox, leading to the following findings:

FF-RQ1) Does switching to RRs affect the number of executed tests per day?

This question analyzes possible correlations between RRs and testing activity. We find that RRs perform more test executions per day, but these tests focus on a smaller subset of the test case corpus.

FF-RQ2) Does switching to RRs affect the number of testers working on a project per day?

To verify the findings of RQ1, we study the number of testers for RRs and TRs. We find that RRs have less testers, but that they have a higher workload.

FF-RQ3) Does switching to RRs affect the number of builds being tested per day?

Apart from number of testers, the number of builds to test also can impact test acticity. We find that RRs test fewer, but larger builds.

FF-RQ4) Does switching to RRs affect the number of configurations being tested per day?

In the same vein as RQ3, we find that RRs test fewer platforms in total, but test each supported platform more thoroughly.

FF-RQ5) Does switching to RRs affect the similarity of test suites or test teams across releases?

Developer and tester retention plays a major role in sustaining an open source system. We find that RRs have higher similarity of test suites and testers within a release series than TRs had.

FF-RQ6) Does switching to RRs affect when the testing happens for a release?

This question checks whether RR systems spread testing activity more uniformly, since they have less time in between releases. Empirical analysis suggests that RR testing happens closer to the release date and is more continuous, yet these findings were not confirmed by the QA engineer.

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.

This paper extends our previous work [11] in two ways. First, we added research questions *FF-RQ5* and *FF-RQ6* that investigate respectively, the impact of RRs on test suite and test team similarity between the releases (i.e. are the same test cases executed in each release and are the same testers performing the testing) , and the timing of test execution for TR and RR releases (i.e., whether testing happens early or late in the release cycle). Having established that rapid releases provide a number of important challenges for testing, we performed a semi-systematic literature study of empirical studies on rapid releases in order to better understand the phenomenon and its origins. The literature review assesses five research questions, with the following answers:

LR-RQ1) Where do RRs originate from?

Our literature study shows that RRs originate from agile, open source and lean software development.

LR-RQ2) What is the prevalence of RRs?

Whereas RRs tend to be associated with less risky software systems, we find that RRs are practiced in several different domains, even in those requiring high reliability.

LR-RQ3) What are the benefits of RRs?

The literature study shows that the main benefits of RRs are shorter time-to-market, rapid feedback and the increased focus of development staff on quality.

LR-RQ4) What are the enablers of RRs?

Key enablers for RR are parallel development, tools for automatic deployment and testing, and the involvement of customers.

LR-RQ5) What are the problems of RRs?

The main problems of RR releases are technical debt, the conflict between high test coverage and high reliability, the customers' willingness to update, and time-pressure.

The rest of the paper is organized as follows. Section 2 provides some background on Mozilla Firefox, while Section 3 describes the design of the Firefox study. The results of this study are presented in Section 4. In Section 5 we analyze the confounding factors and present a theoretical model explaining the relationship between the release model, release length and test effort. Section 6 then discusses the setup and results of the semi-systematic literature review, which is followed by a discussion of the case study and literature review results (Section 7). Finally, Section 8 concludes the paper and outlines some avenues for future work.

2 Research Context

2.1 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 [12], with almost half a billion users. Firefox 1.0 was released in November 2004 and the latest version considered in this study, Firefox 13, was released on June 5, 2012, containing more than 7.5 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 [4,13], which was eroding Firefox's user base. Every TR version of Firefox was followed by an unpredictably 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 a

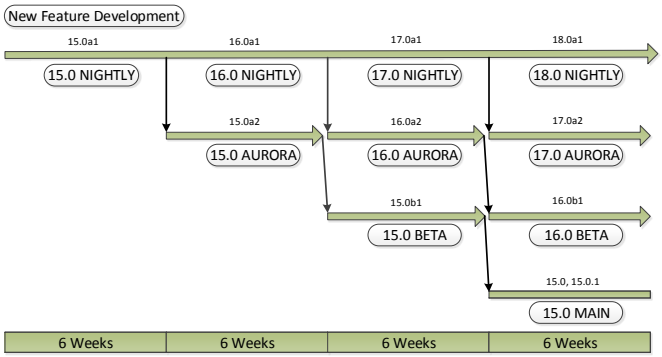


Fig. 1: Rapid release process of Mozilla Firefox.

fixed sequence of four release channels (Figure 1), each of which takes exactly six weeks to complete: nightly, aurora (alpha), beta and main [14]. 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.

2.2 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 [15]. 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 [16]. 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%.

3 Study Design

In order to address the research questions *FF-RQ1* to *FF-RQ6*, 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.

3.1 Data Collection

We performed web crawling of the Litmus system to get the test cases and their execution data 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 consider data only 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 [17]. 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 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 invididual release (alpha, beta, release-candidate, major or minor) each test execution was related to. Hence, we mapped the test execution to the release dates, which were available online [18], using the main release number (2.0-13.0) that was available and the time-stamps.

For each of the analyzed versions, we also extracted the code revision history from the Mercurial repository [19] 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 our empirical findings.

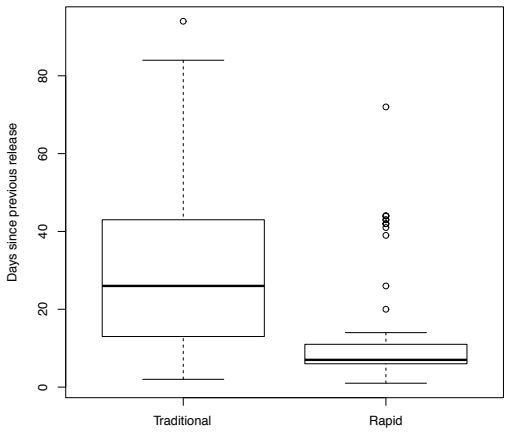


Fig. 2: Distribution of release cycle length (in days) for TRs and RRs.

Table 1: Interpretation of Cliff's delta [20].

Interpretation	Cliff's delta
Small	0.148
Medium	0.330
Large	0.474

3.2 Data Analysis

We analyze the collected data set using a set of metrics defined specifically for each question. For the reader's comfort, 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 [21]), which varies between -1 and 1. Table 1 explains the interpretation of Cliff's delta.

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 [7]. If we consider all types of releases (i.e., alpha, beta, release candidate, major and minor) together, the median time between RR releases is 7 days while it is 26 days for TR releases. This difference is statistically significant with a large effect size, see Table 5.

We did not perform this normalization when analyzing cumulative growth over time of a metric (i.e., Figure 4, Figure 5, Figure 7 and Figure 9), nor in *FF-RQ5*. In the latter question, we use Cohen's Kappa to measure the similarity of test suite and test team between two subsequent releases. Since Cohen's Kappa is similar to a correlation metric (i.e., it has a limited scale from -1 to 1) and it does not increase as the release length increases, there is no need to normalize it for release length.

Cohen's Kappa is originally developed to measure inter-rater agreement between two raters. In our case, the two raters are two subsequent releases, and the agreement is measured across the boolean vector of all test cases (n=1547) and testers (n=6,058), respectively. If two subsequent releases executed exactly the same test cases, Cohen's Kappa value is 1. If only by chance, two subsequent releases execute the same test case, the value is 0. If there is no agreement at all, not even by chance, the value is -1. Cohen's Kappa is a more robust measure of similarity between releases than simply calculating the percentage of different test cases or testers between releases, as it takes into account differences and similarities that occur by chance.

Cohen's Kappa has known problems with unbalanced data (i.e., boolean vectors in which the vast majority is "yes" or "no"), and prior work points out that *"no single omnibus index of agreement can be satisfactory for all purposes"* [22]. For example, let's assume that we have four test cases (i.e., vectors with four boolean values), with the test executions for releases R1 and R2 corresponding to R1=[YYNN] and R2=[YNYN] (Y(es) meaning an executed test and N(o) meaning a not executed test). In this case, Kappa would be 0 because all similarities in the test execution suites are due to chance only. However, imagine that both test case execution vectors would contain one thousand additional legacy test cases that were never executed. Then, vectors R1 and R2 would have much higher similarity due to the thousand no-no agreements and the Kappa between R1 and R2 would increase to 0.499. Obviously, the same thing would happen if there are one thousand yes-yes cases, for example tests that are always executed due to being part of the core of the automated test suite.

To identify whether a Kappa value is affected by this problem, one should follow the suggestion by Cicchetti et al. [23] to calculate proportions of positive and negative similarity p_{pos} and p_{neg}. If these proportions differ substantially from each other, then the Kappa value is biased and the interpretation of the value needs to take this bias into account. Table 2 shows how the test case executions of two releases R1 and R2 can be organized as a 2 X 2 table. In the table, (a) refers to the number of identical test cases that are executed in both releases. Similarly, (b) refers to the number of test cases that are executed in release R2, but not executed in release R1 and so on. Then: p_{pos}=2a/(N+a-d) and p_{neg}=2d/(N-a+d). Thus, when our vectors are R1=[YYNN] and R2=[YNYN], then both p_{pos} and p_{neg} would be 0.5 (see Table 3), indicating that the positive and negative are similar. When we add one thousand zero-zero cases to vectors R1 and R2 for the legacy test cases that are never executed (see Table 4), then p_{pos} is still 0.5 while p_{neg} jumps to 0.999. The latter high value clearly indicates a bias towards negative similarity between test executions in R1 and R2, providing a warning about the causes (or lack) of similarity. In that case, the bias might be one of the possible explanations of the corresponding Kappa value.

Cohen's Kappa ranges from -1 to 1. The generally accepted linguistic labels for values are < 0 to indicate lack of agreement, $0-0.20$ for slight agreement, $0.21-0.40$ for fair agreement, $0.41-0.60$ for moderate agreement, $0.61-0.80$ for high agreement, and $0.81-1$ for very high agreement [24]. The range of p_{pos} and p_{neg} is from 0 to 1, and according to Byrt et al. [22] they should be interpreted similar to specificity and sensitivity. We could not find generally accepted thresholds for p_{pos} and p_{neg}, and although empirical research exists on determining thresholds for the related measures of specificity, sensitivity (recall), or precision [25,26],

no common thresholds exist. Since the purpose of this work is not to determine empirical thresholds, we simply adapt the linguistic labels for Kappa [24]. Since for any large random boolean data set, Kappa is 0 while p_{pos} and p_{neg} are 0.5, we use the following linguistic labels for p_{pos} and p_{neg}: 0.51−0.60 for slight agreement, 0.61−0.70 for fair agreement, 0.71−0.80 for moderate agreement, 0.81−0.90 for high agreement, and 0.91−1.0 for very high agreement.

Table 2: Comparing test case execution similarity of two releases R1 and R2.

		R1		
		Yes	No	Total
R2	Yes	a	b	g
	No	c	d	h
	Total	e	f	N

Table 3: Comparing test case execution similarity of two releases with vectors R1=[YYNN] and R2=[YNYN].

		R1		
		Yes	No	Total
R2	Yes	1	1	2
	No	1	1	2
	Total	2	2	4

Table 4: Comparing test case execution similarity of two releases with vectors R1=[YYNN] and R2=[YNYN] when we add one thousand No-No cases to both vectors.

		R1		
		Yes	No	Total
R2	Yes	1	1	2
	No	1	1001	1002
	Total	2	1002	1004

4 Case Study Results

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

FF-RQ1) Does switching to RRs affect the number of executed tests per day?

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

Table 5: Comparison between metrics for the TR and RR. Effect size uses Cliff's Delta.

	TR (median)	RR (median)	WRST (p-value)	Effect size
Release length	26.00	7.00	**<0.001**	-0.586
#Test exec./day (FF-RQ1)	50.86	127.3	**<0.001**	0.359
#Testcases/day (FF-RQ1)	10.65	14.00	0.855	0.015
#Testers/day (FF-RQ2)	1.667	1.000	**<0.001**	-0.297
#Builds/day (FF-RQ3)	0.427	0.250	**0.004**	-0.242
#Locales/day (FF-RQ4)	0.302	0.143	**<0.001**	-0.356
#OSs/day (FF-RQ4)	0.270	1.286	**<0.001**	0.695
Cohen's Kappa (Test suite) (FF-RQ5)	0.769	0.977	**<0.001**	0.480
Cohen's Kappa (Tester) (FF-RQ5)	0.108	0.625	**<0.001**	0.853
Temporal test distance (FF-RQ6)	0.356	0.200	**<0.001**	-0.467

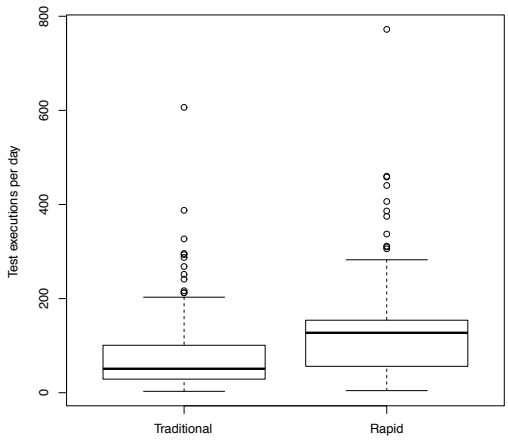
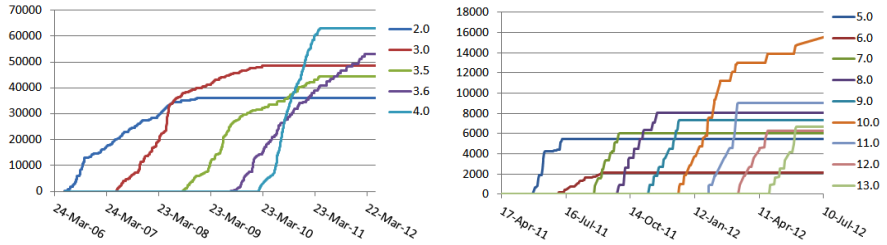


Fig. 3: Distribution of number of test executions per day for TRs and RRs.

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.

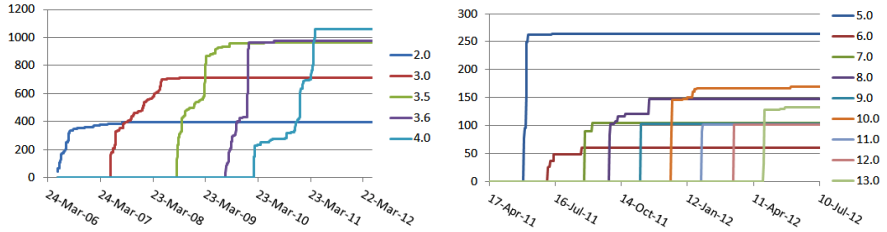
Null Hypotheses: We test the following two null hypotheses to compare the total number of test executions and the number of unique test cases executed for TR and RR models:
H_{01}^1: There is no difference between the number of tests executed per day for RR releases and TR releases.
H_{02}^1: There is no difference between the number of unique test cases executed per day for RR releases and TR releases.
We use the Wilcoxon rank-sum test [27] to test H_{01}^1 and H_{02}^1 using a confidence level α of 1% (*i.e.*, p-value needs to be < 0.01 to reject a null hypothesis).

Metrics: For each alpha, beta, release-candidate, major and minor version of Firefox in our data set, we compute the following two metrics: the number of tests executed and the number of unique test cases executed. The number of tests executed captures the amount of testing performed per release. The number of unique test cases executed captures the functional coverage of the tests per release.



(a) (b)

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



(a) (b)

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

Functional coverage is the degree to which the different features in a software version are tested by a test suite. Our only measure of functional coverage is the number of unique test cases executed (the more functionally different test cases are being executed, the higher the functional coverage). Other possible measures of functional test coverage could be for example requirements coverage or use case coverage.

- #Test exec./day: the number of tests executed per day.
- #Test cases/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 5 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 release 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) [28], i.e., it is meant to last for 54 weeks instead of 6 (lifetime of 9 "normal" RRs). An ESR helps corporate clients [29] 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 then 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_{02}^{1}. This means that, although a higher absolute number of unique test cases gets executed in TR, there is no difference between release models when the shorter time frame for RR releases is taken into account.

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*

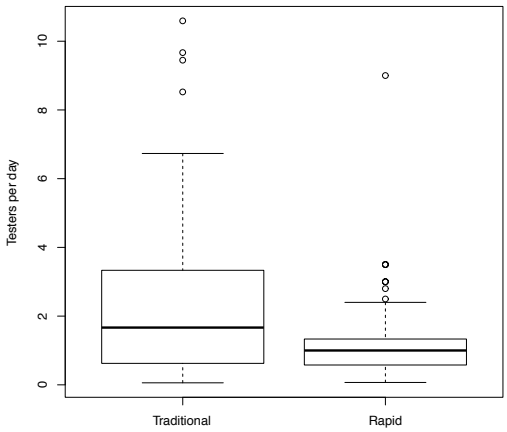
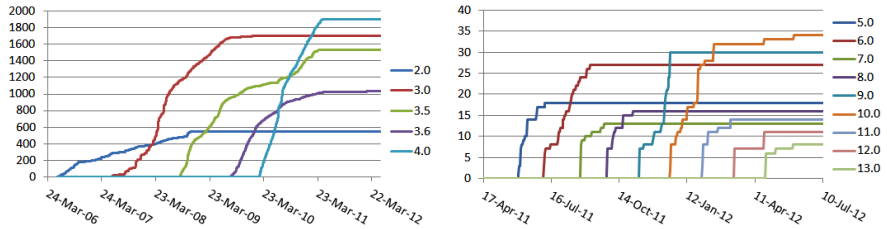


Fig. 6: Distribution of number of testers per day for TRs and RRs.



(a) (b)

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

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.*

FF-RQ2) Does switching to RRs affect the number of testers working on a project per day?

Motivation: With short release cycles, development teams have less time to implement and test new features before they are released to users. In **FF-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?

Null Hypothesis: We test the following null hypothesis to compare the number of testers for TR and RR releases:
H_{01}^2: There is no difference between the number of testers for RR releases and TR releases.
Similar to **FF-RQ1**, we use the Wilcoxon rank-sum test [27] to test H_{01}^2 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/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_{01}^2. 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 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 instructions), perhaps due to the automation breaking down. However, if changes in the share of test automation would have dramatically impacted testing, this should have led to a significant increase in the number of test executions (or cases) per day over time. Section 5 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*

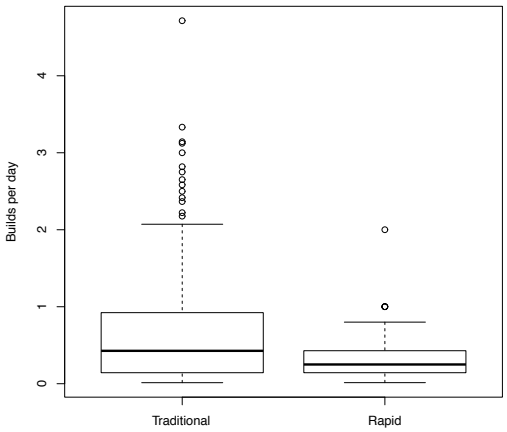


Fig. 8: Distribution of the number of builds tested per day for TRs and RRs.

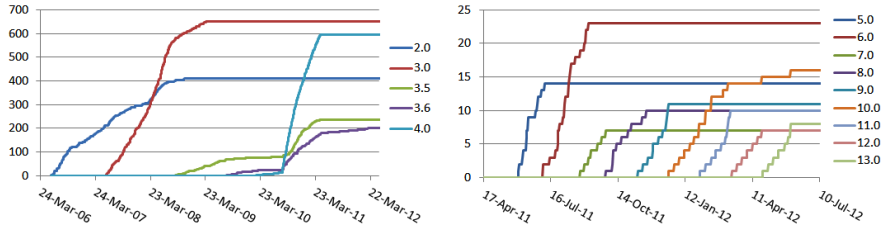
faster through contractors than through hiring. The time afforded to us to make the switch to rapid releases left little room for failure which is why we took that approach.". The number of testers has also been impacted by some competing Mozilla projects. Regarding test automation, the QA engineer noted 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]"*.

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

FF-RQ3) Does switching to RRs affect the number of builds being tested per day?

Motivation: The finding that more tests are executed per day for RR releases 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 builds to be tested. 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_{01}^{3}: There is no difference between the number of tested builds per day for RR releases and TR releases.
H_{02}^{3}: There is no difference between the number of commits per day for RR releases and TR releases.



(a) (b)

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

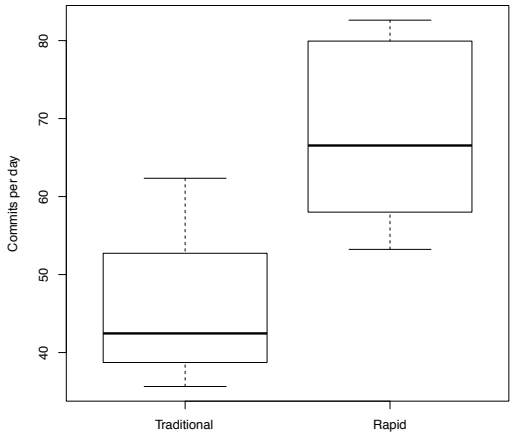


Fig. 10: Distribution of the number of commits per day for TRs and RRs.

We again use the Wilcoxon rank-sum test [27] to test these null hypotheses using a 1% confidence level.

Metrics: We calculated the following metrics:

- #Tested builds/day: the number of tested builds per day.
- #Commits/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 releases, since for versions developed following the TR model, commits are hard to link to specific alpha, beta, release-candidate or minor 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}, i.e., the higher relative frequency of test executions cannot be explained by more builds being tested. Instead, it seems that each build is tested more thoroughly (albeit with a smaller coverage, see **FF-RQ2**).

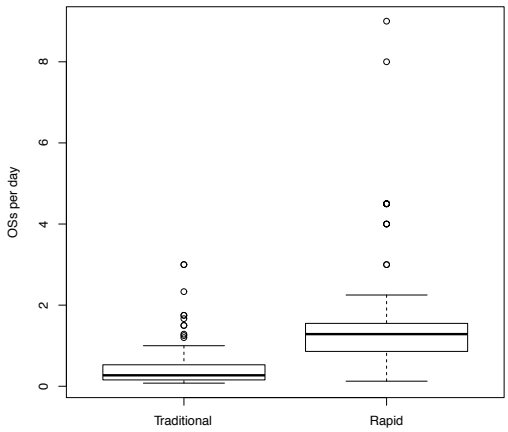


Fig. 11: Distribution of number of operating systems tested per day for TRs and RRs.

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}.

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, although 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.*

FF-RQ4) Does switching to RRs affect the number of configurations being tested per day?

Motivation: In this research question, we study the different configurations that are tested, such as different operating systems or support for more locales (i.e., language settings and internationalization [30]). More thorough testing of different configurations could explain the larger number of tests per build (**FF-RQ3**), as well as the higher workload of individual testers (**FF-RQ2**).

Null Hypotheses: We test the following null hypotheses:
H_{01}^{4}: There is no difference between the number of tested locales per day for RR releases and TR releases.
H_{02}^{4}: There is no difference between the number of tested operating systems per day for RR releases and TR releases.

We again use the Wilcoxon rank-sum test [27] to test these null hypotheses using a 1% confidence level.

Metrics: We calculated the following metrics:

- #Tested locales/day: the number of tested locales per day.
- #Tested operating systems/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 manually on only one locale.** 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"*. This confirms our findings about uniform test attention.

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

FF-RQ5) Does switching to RRs affect the similarity of test suites or test teams across releases?

Motivation: This question investigates how rapid releases affect test suite and test team similarity between releases. A more diverse test suite is preferred (less similarity) as it has a higher likelihood of finding defects, and the same holds for a more diverse set of testers [31]. However, in **FF-RQ1** we learnt that the smaller

Table 6: Average Kappa, p_{pos} and p_{neg} of test case similarity for successive pairs of main releases only.

TR releases	Kappa	p_{pos}	p_{neg}	RR releases	Kappa	p_{pos}	p_{neg}
2.0 vs. 3.0	0.27	0.51	0.73	5.0 vs. 6.0	0.24	0.28	0.91
3.0 vs. 3.5	0.30	0.67	0.61	6.0 vs. 7.0	0.27	0.31	0.93
3.5 vs. 3.6	0.91	0.97	0.94	7.0 vs. 8.0	0.61	0.65	0.95
3.6 vs. 4.0	0.28	0.76	0.52	8.0 vs. 9.0	0.80	0.81	0.98
				9.0 vs. 10.0	0.67	0.70	0.97
				10.0 vs. 11.0	0.71	0.74	0.97
				11.0 vs. 12.0	1.00	1.00	1.00
				12.0 vs. 13.0	0.85	0.86	0.99
average	0.441	0.726	0.702	average	0.644	0.670	0.964

set of RR test cases consists of a fixed part and a variable part depending on the feature set of a new release, while in **FF-RQ2**, we saw that a smaller team of testers (including a fixed team of contractors) had become responsible for RR testing. Here we analyze whether the same tests are being tested between every two subsequent TR or RR releases, and whether this is done by the same testers.

Null Hypotheses: We test the following null hypotheses:
H_{01}^5: There is no difference between the similarity of test suites executed for RR releases and TR releases.
H_{02}^5: There is no difference between the similarity of test teams for RR releases and TR releases.
We again use the Wilcoxon rank-sum test [27] to test these null hypotheses using a 1% confidence level.

Metrics: As explained in Section 3.2, this question uses Cohen's Kappa to measure similarity between the choice of test cases and testers of two subsequent releases within a single main release. Additionally, we use p_{pos} and p_{neg} to investigate whether the similarity is due to positive or negative recurrence. In particular, we calculate the following metrics:

- #Similarity of test suites: the Kappa value between the test cases executed in two subsequent releases.
- #Similarity of test teams: the Kappa value between the testers in two subsequent releases within a main release.
- #Positive similarity of test cases: the p_{pos} value between the test cases executed in two subsequent releases within a main release.
- #Positive similarity of testers: the p_{pos} value between the testers in two subsequent releases within a main release.
- #Negative similarity of test cases: the p_{neg} value between the test cases executed in two subsequent releases within a main release.
- #Negative similarity of testers: the p_{neg} value between the testers in two subsequent releases within a main release.

Note that these measures are different from the number of test cases executed per day of **FF-RQ1**, or the number of testers executing test cases per day of **FF-RQ2**. Those metrics only concern individual releases, while the similarity measures of this question look at the difference between the choice of test cases and testers of two subsequent releases.

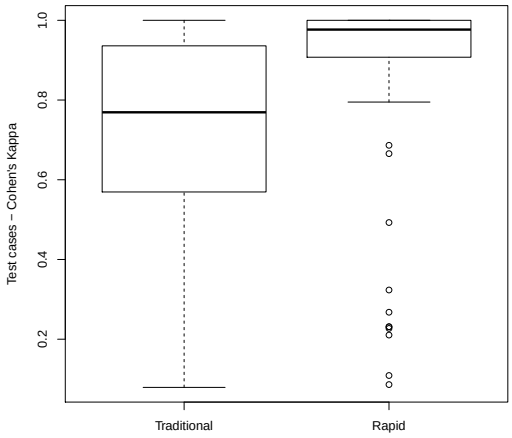


Fig. 12: Similarity (Cohen's Kappa) of executed test cases (test suites) between two subsequent RR or TR releases.

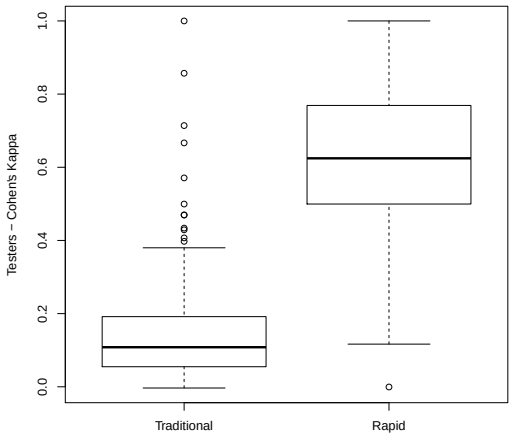
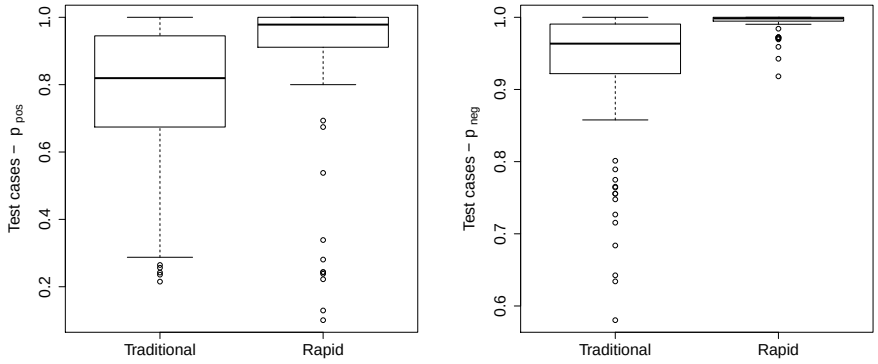


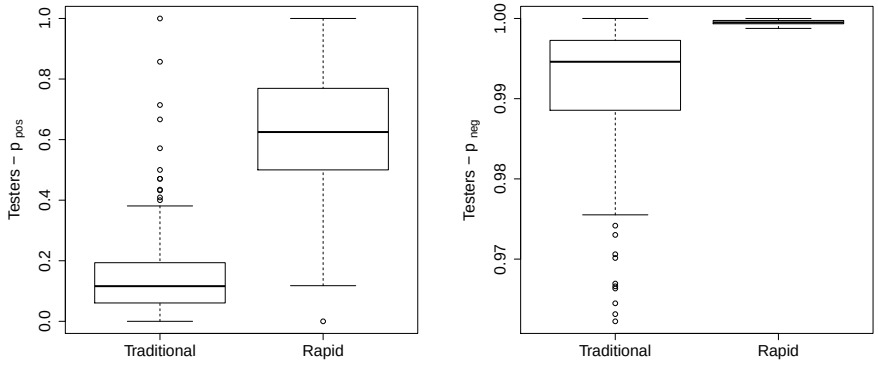
Fig. 13: Similarity (Cohen's Kappa) of testers between two subsequent RR or TR releases.

RR releases have higher test suite and test team similarity between individual releases. Test suites between releases have high similarity in RR releases, with median Kappa of 0.977, see Figure 12. Although TR releases also have a relatively high similarity (median Kappa of 0.769), this is significantly lower than for RR releases, thus we reject H_{01}^{5}. Studying Figure 14 (a) and (b)



(a)

(b)



(c)

(d)

Fig. 14: Proportions of positive p_{pos} and negative p_{neg} [23] similarity between two subsequent RR or TR releases, for tests ((a) and (b)) and testers ((c) and (d)).

shows that both the positive (p_{pos}) and negative (p_{neg}) similarity are higher in RR and both differences are also statistically significant ($p<0.001$). Positive similarity measures the proportion of test cases that are executed in two subsequent releases and negative similarity measures the proportion of test cases that are executed in neither of two subsequent releases. The investigation of p_{pos} and p_{neg} provides strong support that the higher test suite similarity in RR is trustworthy and not due to unbalanced data.

Since we did not expect such high similarities, we also checked the similarity of test suites at the level of successive pairs of main releases, i.e., where we group all individual releases into their main release. For RR releases, Table 6 shows that the average Kappa over the main releases was 0.644, while for TR releases the average

Table 7: Average Kappa, p_{pos} and p_{neg} of tester similarity for successive pairs of main releases only.

TR releases	Kappa	p_{pos}	p_{neg}	RR releases	Kappa	p_{pos}	p_{neg}
2.0 vs. 3.0	-0.02	0.12	0.80	5.0 vs. 6.0	0.22	0.23	0.99
3.0 vs. 3.5	-0.20	0.12	0.67	6.0 vs. 7.0	0.55	0.56	1.00
3.5 vs. 3.6	-0.14	0.09	0.76	7.0 vs. 8.0	0.60	0.60	1.00
3.6 vs. 4.0	-0.15	0.10	0.71	8.0 vs. 9.0	0.36	0.36	1.00
				9.0 vs. 10.0	0.30	0.30	1.00
				10.0 vs. 11.0	0.47	0.47	1.00
				11.0 vs. 12.0	0.67	0.67	1.00
				12.0 vs. 13.0	0.67	0.67	1.00
average	-0.130	0.108	0.733	average	0.479	0.482	0.997

Kappa over the main releases was 0.441. This lower test suite similarity at the level of main releases (in contrast to individual releases) is what one would expect, since each main release focuses on a different scope of features and we have seen in **FF-RQ1** that a release's test suite (partially) depends on the release's scope. The positive test suite similarity is somewhat higher in TR than in RR at the main release level, which contradicts our findings from individual releases. However, one should be careful interpreting the averages of Table 6, as the variations in the metrics are large and the data set at the level of main releases is too small for reliable conclusion.

Figure 13 shows that the similarity of test teams between releases is higher for RR, with a median Kappa of 0.625 vs. 0.108 for TR releases. This difference is statistically significant and we therefore reject H_{02}^5. Studying Figure 14 (c) and (d) shows that both the positive and negative similarity of test teams are higher in RR and both differences are also statistically significant ($p<0.001$).

The result of **FF-RQ2** already showed that there is a reduction in number of testers, while the result of **FF-RQ5** shows that mostly the same testers work in all releases. These are two different findings. Indeed, having a smaller test team in the worst case could mean that tester retention is very hard, with each release seeing a different group of testers starting up. Our findings suggest that this is not the case, and that despite its smaller size the composition of testers remains more stable over time.

To understand whether the more stable composition is due to testers being stable inside a particular release series (e.g., all alpha, beta and main releases of Firefox 5.x) compared to across different release series (e.g., Firefox 5.x vs. 6.x), we also calculated test team similarity at the main release level (Table 7). We find that tester similarity between the main releases is lower than between the individual releases. This suggests that, similar to test cases, testers are relatively stable inside a particular release series, while across different release series the composition of the test team varies. This might be due in part to the reduction in the testing community, as discussed in **FF-RQ2**.

Feedback QA engineer Regarding the differences in similarity of test suites, the QA engineer explained that before the switch to rapid release, tests were grouped into basic smoke tests run once per week, more complex tests run once per beta version, and very deep and hard to set up tests run right before the main release. All tests in the first category and many of the tests in the second were automated when switching to RRs.

After the switch, Litmus and Moztrap (successor of Litmus) tests consist of a varying set of feature tests that change for each main release, a fixed set of smoketests run on a weekly basis *"as a community-oriented set of tests to teach newcomers about testing"*, and a more or less fixed set of regression tests run throughout the beta releases. As a rough ballpark number, the QA engineer estimated *"This is a really really wild guess but it might breakdown something like 10:10:80 per 100 tests (Smoketests, Regression Tests, and Feature Tests respectively)"*. Regardless of the exact percentage, it is clear that the main release-specific set of tests dominates, which explains why similarity across release series is relatively low compared to between the individual releases.

Regarding test team similarity, there are three main groups of testers. Test leads are assigned to a specific version that they follow from start to end, across the different release channels. Afterwards, they start with a new version from the first channel on. These test leads come from a fixed pool of people. The second group (again a relatively fixed pool of people) consists of testers that are in charge of specific features, regression testing and bug triaging (on a weekly basis). Contrary to the test leads, *"These people are working across all four branches at any given time. Once their feature is shipped they pick up a new feature that's under development in Nightly"*. Finally, volunteers help out with smoke testing and bug triaging, on-demand (as requested by a test lead). This group consists of a fixed group of committed people and ever-changing group of newcomers. The QA engineer noted that *"There is certainly a group of people who come to report a bug and leave once they've tested its been fixed but I think that group is in the minority"*. These findings support our finding that for RR, the set of testers is relatively fixed within a release series and explains that the changes in testers across release series are caused by some testers being tied to one particular release or feature only, and by a changing group of newcomers.

> *RR releases have higher similarity in test suites and test teams inside a release series.*

FF-RQ6) Does switching to RRs affect when the testing happens for a release?

Motivation: This final research question further explores our findings for **FF-RQ1**, where we found that RR releases execute more tests per day than TR releases. Although we analyzed the cumulative trend of the number of test executions over time (Figure 4), we could not differentiate between beta, minor or major releases, nor did we analyze or compare for each release whether testing occurs early on, is done at the last minute or at a constant rate during the release cycle. This is important to understand, since prior work indicates that early testing has a positive impact on product quality [9]. Hence, here we study the differences in the temporal distance of testing, i.e., does system testing happen early or late in the release cycle.

Null Hypothesis: We test the following null hypothesis:
H_{01}^{6}: There is no difference between the normalized temporal test distance of RR releases and TR releases.
We again use the Wilcoxon rank-sum test [27] to test this null hypothesis using a 1% confidence level.

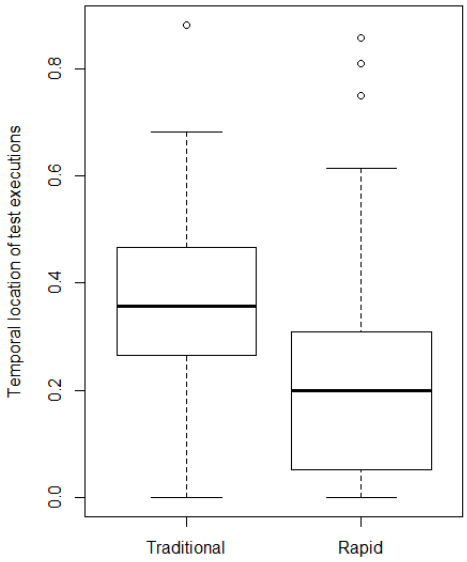


Fig. 15: Normalized temporal distance of test executions. Values closer to zero indicate that testing happens closer to the release date.

Metrics: Our measure for this hypothesis is the temporal distance between a test case execution and the release date. All test cases that are executed on the day of the release get a value of zero. The test cases executed the day before get a value of one and so on. To summarize these values into one number, we calculate the average value across all test cases, then divide by the release cycle length to normalize the average to a value between zero and one (to resolve differences in release cycle length). For each release, we obtain the following metric:

– Normalized temporal test distance: the average distance of test cases to the release date, normalized by the release cycle length of that release.

RR testing happens closer to the release date. Figure 15 shows that the bulk of RR testing happens closer to the release date. This result is statistically significant (0.200 vs. 0.356), hence we reject H_{01}^{6}. This suggests that testing in RR releases perhaps is performed under greater time or deadline pressure than in TR, although **FF-RQ1** showed that in RR releases testing is rather linearly distributed, starting roughly when the previous main release has been released (Figure 4).

To further investigate the effect of deadline pressure and the timing of the testing, Figure 16 plots a four week (28 days) moving average of the number of test executions of all releases for the TR and RR releases. This means that for each day, we calculate the average number of test executions in the past four weeks (28 days), which gives us a measure of the amount of test effort in the near past. The

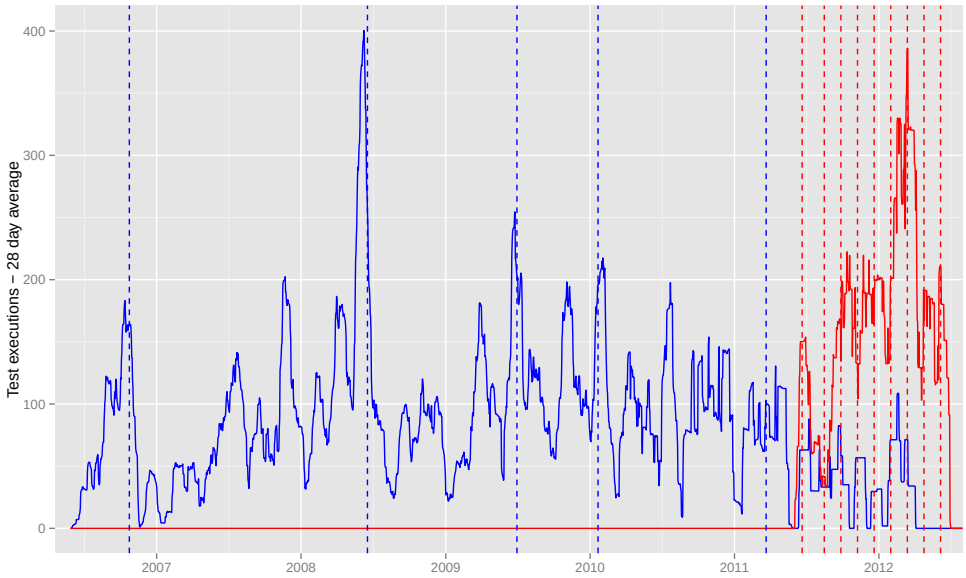


Fig. 16: 28-day moving average of the number of test executions for all TR (blue) and RR releases (red). Dashed lines represent main release dates for TR (blue 2.0, 3.0, 3.5, 3.6, 4.0) and RR (red 5.0-13.0)

figure shows that spikes in testing, that correspond to the main releases, have equal heights in TR and RR. Additionally, the figure shows that the higher number of test execution of RR are due to having continuously high volume. Thus, in TR the spikes in the amount of testing are proportionally higher than in RR due to lower overall test execution volume. The extra high peaks in early 2012 correspond to the ESR version 10.0. During that time, the upcoming release 11.0 was tested while the testing for the released ESR version 10.0 continued. To conclude, RR testing sees a more continuous activity with less variation.

Feedback QA engineer Regarding the more deadline oriented testing in RR the QA engineer was unsure *"I don't know, I think testing has always been and will always be deadline oriented"*. This observation matches the spikes in Figure 16 for testing related to the main releases of both TR and RR. However, we received no confirmation of our findings in Figure 15 that individual RR releases would be more deadline-oriented. Perhaps, this observation, which comes from 213 individual releases, is unnoticeable in practice. Additionally, the interview confirmed that after the switch to rapid release, testing continuously is done in a high volume, in the following sense: *"Before the switch to rapid release the deadlines ran in serial, now they run in parallel. As such we now test in parallel (or near parallel). Betas get most of our attention as they're the closest to release, Aurora is our second priority, and Nightly is our third priority"*. However, the QA engineer pointed out that periods with very low official testing volume in TR, in particular the zeros in Figure 16, might have masked other means of testing not visible in the Litmus system: *"When builds were in Alpha before rapid release there was daily dogfooding"*.

Table 8: Kendall'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 9: Partial correlation of three variables (release model, length and date) with test effort, while controlling for two out of the three variables. Significance levels *=0.05 **=0.01, ***=0.001.

FF-RQ	Partial Kendall correlation coefficients		
	Release Model	Release Length	Project Evolution
#Test executions/day (FF-RQ1)	0.026	-0.414***	0.048
#Test cases/day (FF-RQ1)	-0.231***	-0.593***	0.017
#Testers/day (FF-RQ2)	-0.224***	-0.331***	-0.069
#Builds/day (FF-RQ3)	-0.105*	-0.258***	-0.176***
#Locales/day (FF-RQ4)	-0.281***	-0.443***	-0.115*
#OSs/day (FF-RQ4)	0.149**	-0.822***	0.130**
Cohen's Kappa (Test suite) (FF-RQ5)	0.187***	0.117*	0.177***
Cohen's Kappa (Tester) (FF-RQ5)	0.422***	-0.036	0.030
Temporal test distance (FF-RQ6)	-0.176***	-0.002	-0.110*

Table 10: Effects of Rapid Releases.

	Significant increase	**No Significant effect**	**Significant decrease**
Effect of RR	Test executions, Operating systems, Test case similarity, Tester similarity	Unique test cases	Release length, Testers, Builds, Locales, Temporal test distance
Effect of RR after controlling for confounders	Operating systems , Test case similarity, Tester similarity	Test executions	Release length, Testers, Builds, Locales, Temporal test distance, Unique test cases

Our findings suggest that RR testing is more deadline-oriented and continuous, yet this was not confirmed by the QA engineer. The spikes in the amount of TR testing before main releases are proportionally higher than for RR testing.

5 Confounding Factors and a Theoretical Model

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 choice of release model itself, but due to the release cycle length, since some of the TR releases have a shorter release cycle as well

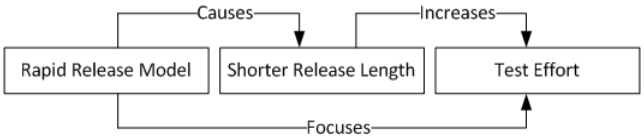


Fig. 17: Model explaining the relationship between release model, release length and test effort.

(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 8). In these calculations, release length (days between releases) and project evolution (the time-stamp of each release date) are simply modelled as continuous numeric variables, while release model is nominal, set either to zero for TRs or one for RRs.

To analyze the impact of these confounding factors, we investigated the effect of release model, release length, and project evolution on the metrics calculated for the research questions, while controlling the confounding effect that these variables have on each other. For this, we used partial correlations (R package ppcor [32]), in which the correlation of one of the FF-RQ's metrics between RRs and TRs is measured, while controlling for two variables out of release model, release length, and project evolution. We used the non-parametric Kendall tau correlation instead of linear multiple regression, since the data is not normally distributed.

Table 9 shows that the release model has a significant effect for eight out of the nine metrics when controlling for the release length and project evolution. It appears that the number of test cases, testers, 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. Furthermore, test suite and test team similarity increase with RR model, while temporal test distance (early testing) shrinks, making the testing more deadline-oriented. 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 **FF-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 an earlier release. 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.

When looking at project evolution, we find five significant correlations. It appears that the number of builds and locales tested decreases as the project moves

along while the number of tested operating system increases. Perhaps this is due to project members learning what variation factors in testing are more significant than others. This seems to be confirmed by the fact that test suite similarity increases, perhaps due to increased understanding about which test cases are important. Finally, testing starts later as time moves along. This might be caused by the use of a more similar set of test cases: testing a familiar set of test cases takes less time, thus, one starts working on it later. However, these findings regarding project evolution are hypotheses and should be more thoroughly investigated in further studies.

Taking all of this into account, our interpretation of the relation between release model and test effort is depicted in Figure 17. 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 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.

Table 10 shows the effects of rapid releases before and after controlling for the confounding factors. We like to think that the first row of the table shows the impact of RR from a practitioner's viewpoint. Practitioners are not interested whether something stems from changes to release length or to the rapid release model as these two are highly correlated, as seen in Figure 2 and Table 8. However, from a researcher or developer viewpoint it is interesting to see whether something is explained directly by the release length or by the change in the release model, cf. Table 9. Additionally, the separation allowed us to create a model for rapid release testing that explains the effects of the shortening of release cycle length and the necessary level of test effort, see Figure 17.

6 Semi-Systematic Literature Review

Given the finding that, even after controlling for confounding factors, rapid releases have an important impact on system testing, we decided to analyze the concept of rapid releases from closer by. This section discusses our semi-systematic literature review on the migration from TR to RR release models. Our literature review fullfills some criteria of a systematic literature review (SLR) [33], but not all. In particular, the SLR criteria that were fulfilled are design and documentation of search strings, usage of dedicated databases, analysis of number of hits, documenting the included papers for each search string, and the usage of explicit inclusion/exclusion criteria. The SLR criteria that we failed to meet are having multiple authors evaluate the papers, and doing more rigorous quality assessment of the study, data extraction and data synthesis. Additionally, we included papers that were not found via our search strings, which is not part of SLR process. We therefore called our literature review a semi-systematic literature review, since this seems to be the most accurate description.

In particular, we searched for existing empirical studies of rapid releases in software engineering, to answer the following research questions:

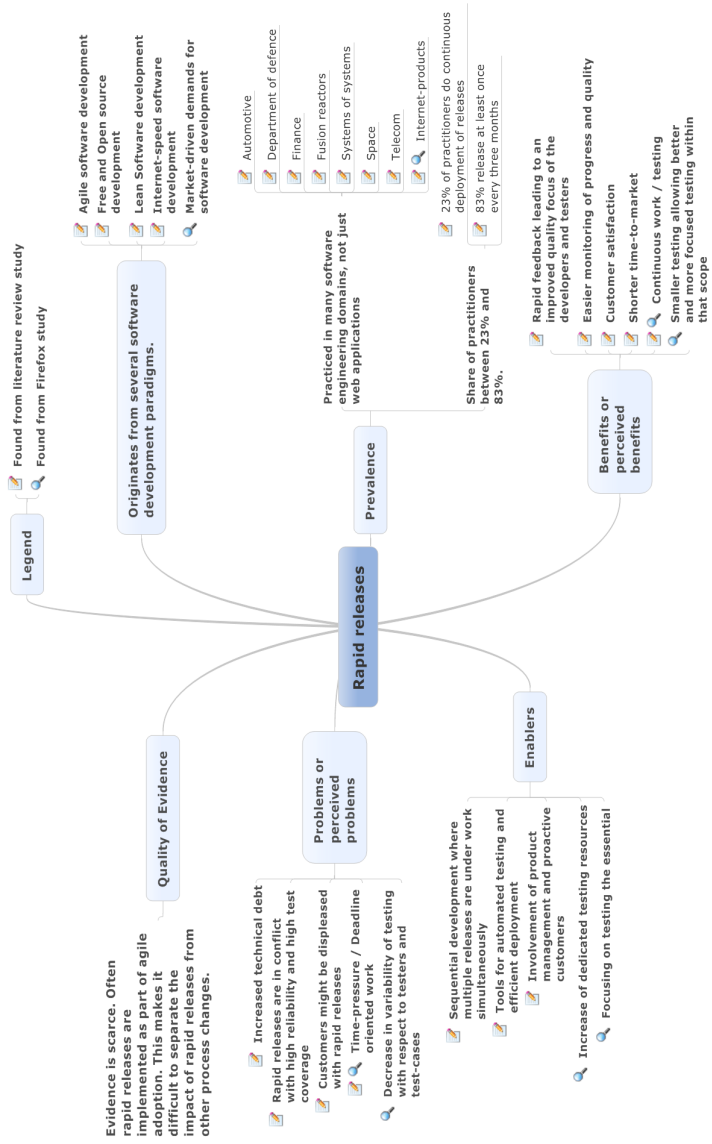


Fig. 18: Main findings of the semi-systematic literature review and the Firefox study.

LR-RQ1) Where do RRs originate from?

LR-RQ2) What is the prevalence of RRs?

LR-RQ3) What are the benefits of RRs?

LR-RQ4) What are the enablers of RRs?

LR-RQ5) What are the problems of RRs?

Table 11 describes the search queries that were used and the number of papers returned for each of them in paper abstracts hosted by the Scopus database. The queries are variations of the terms "continuous release", "continuous delivery" and "continuous deployment" [34]. The Scopus database contains the titles and abstracts of articles of all major academic databases, such as IEEEXplore, ACM digital library, Elsevier and Springer.

Once we had obtained the results of these queries, we applied our inclusion and exclusion criteria. To be included paper needed to fulfill one or more of the following criteria:

- Papers with the main focus on rapid releases
- Papers with the main focus on an aspect of software engineering that was largely impacted by the rapid release model
- Papers with the main focus on an agile, lean or open source process having results of rapid releases

From the included papers, we particularly excluded:

- Opinion papers without empirical data on rapid releases

Table 11 shows how our systematic search was plagued by a low signal to noise ratio, i.e., we found many papers that mentioned our search terms, but lacked actual empirical data about rapid releases in an industrial or open source software engineering context. In many abstracts, rapid releases were mentioned, but no empirical results were given in the paper. Conversely, we encountered papers where rapid releases were not mentioned in the abstract, but the papers still presented results of rapid releases in the other sections. To deal with those cases, we performed additional literature search in a less systematic approach, focusing on papers that we were already aware of ourselves, papers found by a full text search using google scholar, and papers studying agile adoption that did not explicitly mention our search terms in their abstract. We believe that our search results offer the most comprehensive review of rapid releases to date.

The following sections present our findings in more detail, together with the appropriate references. Figure 18 provides the reader with a high-level overview of the results from our semi-systematic literature review.

6.1 Context and Prevalence of Rapid releases - LR-RQ1 and LR-RQ2

Rapid releases are becoming an important part of several software engineering processes, and they are also used in several different software domains. Thus, rapid releases are cross-cutting approaches that are highly relevant to many software development organizations. At the same time, the amount of high quality evidence, i.e., the measured impact of rapid releases is weak. We can find papers on agile adoption, but often the papers rely on perceived opinions, in the form of interviews or survey answers, rather than actually measured data. Additionally, the agile adoption papers do not make a clear distinction between TR and RR releases, which makes it difficult to see what changes are caused by rapid releases.

Table 11: Search queries of the literature review.

Search String used in Scopus	Papers included / Papers found	Included papers
TITLE-ABS-KEY((“continuous release” OR “rapid release” OR “frequent release” OR “fast release” OR “quick release” OR “speedy release” OR “accelerated release” OR “agile release” OR “short release” OR “lightning release” OR “brisk release” OR “hasty release” OR “compressed release” OR “release length” OR “release size” OR “release cadence”) AND Software) AND (LIMIT-TO(SUBJAREA,“COMP”) OR LIMIT-TO(SUBJAREA,“ENGI”) OR LIMIT-TO(SUBJAREA,“BUSI”))	15/72	[35,36,37,38, 39,40,41,42,7, 43,44,45,46, 47,48]
TITLE-ABS-KEY((“continuous delivery” OR “rapid delivery” OR “frequent delivery” OR “fast delivery” OR “quick delivery” OR “speedy delivery” OR “accelerated delivery” OR “agile delivery” OR “short delivery” OR “lightning delivery” OR “brisk delivery” OR “hasty delivery” OR “compressed delivery” OR “delivery length” OR “delivery size” OR “delivery cadence”) AND Software AND Delivery) AND (LIMIT-TO(SUBJAREA,“COMP”) OR LIMIT-TO(SUBJAREA,“ENGI”) OR LIMIT-TO(SUBJAREA,“BUSI”))	4/145	[49,50,51,52]
TITLE-ABS-KEY((“continuous deployment” OR “rapid deployment” OR “frequent deployment” OR “fast deployment” OR “quick deployment” OR “speedy deployment” OR “accelerated deployment” OR “agile deployment” OR “short deployment” OR “lightning deployment” OR “brisk deployment” OR “hasty deployment” OR “compressed deployment” OR “deployment length” OR “deployment size” OR “deployment cadence”) AND Software AND Deployment) AND (LIMIT-TO(SUBJAREA,“COMP”) OR LIMIT-TO(SUBJAREA,“ENGI”) OR LIMIT-TO(SUBJAREA,“BUSI”))	5/211	[35,53,54,55, 56]

6.1.1 Software Development Processes - LR-RQ1

First, shorter release cycles form an integral part of various agile methodologies, such as XP [3] or Scrum [57], since they enable companies to act faster on changing market demands and they enable faster customer feedback. The latter is believed to lead to better product quality, since every release undergoes separate QA scrutiny and customers will quickly report severe bugs, which then in turn can be fixed earlier.

Second, the free and open-source software community (FOSS) has also been propagating a rapid release approach, popularized by Raymond’s book using the statement “Release early. Release often.” [31]. The release cycles of FOSS development have been studied in various papers. Ten years ago, Zhao et al. found that 54% of the open source apps released at least once per month. Five years later, Otte et al. [58] found slightly contrasting numbers (on a different set of apps), *i.e.*, 49.3% released at least once per 3 months. Although this is still relatively rapid, it is not clear why this shift has happened. It seems that projects still followed shorter cycles, but they seemed to have moved to a combination of time- and feature-based releases instead of time-based only.

Third, lean software development [59,60,61,62], which originates from Toyota’s car manufacturing philosophy, emphasizes rapid releases. A cornerstone of lean is

a continuous flow of work items and a minimized waste and inventory. In software engineering, the inventory could be features or bug fixes that have not yet been delivered to the customer. Thus, a lean approach to software engineering also drives rapid releases. For example, a case study in the UK [50] shows how the adoption of the lean approach resulted in more rapid releases with shorter lead time.

Fourth, rapid releases are also known under the terms "internet-time" software development or "high-speed" development. For example, Cusuman et al. (p. 233, [63]) show how major releases of Netscape and Microsoft browser and server products occurred faster than the operating system releases of Microsoft of that era (3 to 18 months versus 24 to 41 months). Yet, even the browser and server products release cycle was relatively slow in today's standards. Additionally, a case study of high-speed development in ten US companies [36] points out that frequent software releases are a key practice in high-speed software development. The major finding regarding software quality was to have frequent synchronizations of work through builds and have stabilization periods for more serious testing. Those studies present no measures of the impact of rapid releases on software quality, software testing, or other software engineering concepts.

6.1.2 Domains - LR-RQ2

In recent years, many modern commercial software projects [64,65] and open source projects backed by a company [13,66] have switched towards shorter release cycles. Tool builders and researchers (e.g., [8]) have focused especially on enabling continuous delivery [34]. Amazon, for example, deploys on average every 11.6 seconds [65], achieving more than 1,000 deployments per hour.

We found studies of rapid releases from several domains, such as finance [53, 64], automotive [46,54], telecom [9], online shops [65], and systems of systems from the surveillance domain [55]. In addition, even domains where strict processes are traditionally followed, such as the US Department of Defense [41], fusion reactors [42] and space shuttles [39] are starting to use rapid releases. Hence, rapid releases can be part of any software development domain. However, no concrete findings or numbers are provided to prove why rapid releases and related ideas improve on the existing approaches, making RR models still a largely open area.

6.1.3 Prevalence and Importance - LR-RQ2 and LR-RQ3

We found several survey studies that have assessed the prevalence of rapid releases. A survey of agile adoption and perceptions at Microsoft [38] found that rapid release was the second most mentioned benefit of agile software development and it was practiced by 2/3 of the respondents. It was perceived that rapid releases enable easier progress tracking, easier monitoring of the quality of the software, more rapid feedback to improve the product, a reduction of turnaround time and hence easier bug fixing.

Another survey of agile perception in 62 UK organizations found that 53% of these organizations delivered software at least once a month, 31% used three month cycles while only 17% delivered just the final release [52]. Kong et al.'s survey with 57 developers [67] shows that shorter release cycles are (amongst others) the best practice for "achieving customer satisfaction" and for "being responsive to customers' changing needs". In contrast, the 2012 State of Agile Survey [68] with

4,048 participants showed that shorter releases (in the survey the question asked for use of continous deployment and not rapid releases) are the least popular agile practices, mentioned by 23% of the respondents. Still, 73% of the people stated that reduced time to market was ranked as the number one reason for agile adoption.

Although some contradictions between the results can be found, due to the differences in populations and questionnaire setup, still it seems that rapid releases are a highly popular industrial practice that is often among the top items among agile practices.

6.2 Rapid releases as main study targets - LR-RQ3 and LR-RQ5

Despite the prevalence and increased adoption, 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. However, proportionally less defects were fixed (when normalizing for the shorter time between releases) and users experienced crashes sooner than with traditional releases.

Baysal et al. [37] found no significant difference in source code patch life cycle when comparing the traditional and rapid releases of the Mozilla Firefox project. Thus, that part of development process was unaffected by the move to the rapid release model.

Lavoie et al. [43] studied Firefox source code changes in traditional and rapid releases with a clone detector. They found that in rapid releases there are fewer changes between the main version releases at the source code level. However, the changes were not normalized for the time elapsed between releases and the authors themselves speculate that "... even if the release cycle changed, the total amount of work put in the same period of time may still be similar".

Maalej [45] found that individuals working in rapid release projects (with a release cycle of less than 4 weeks compared to a control group with a release cycle longer than 8 weeks) preferred software engineering tool integration through tasks whereas the control group preferred an activity- or component-based tool integration. This suggests that rapid releases increase work orientation to tasks over software engineering process or software architecture, e.g., what steps I have to take to get this task done over what steps I have to take in software design, implementation or testing.

6.3 Benefits of rapid releases in agile and lean process adoption - LR-RQ3

A number of case studies have been published by companies that moved to agile methodologies [9,10,69,70]. However, since many agile techniques and team re-

structurings were introduced at once, the observed changes cannot be related to shorter release cycles alone.

Petersen et al. [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 also reported on the improvements of test coverage at unit-test level. Earlier research [71] 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.

Escrow.com reduced its release cycle to iterations of 2 weeks, with up to three iterations forming a customer release [69], resulting in a reduction of the number of defects by 70%. They noticed that the number of defects surviving until acceptance testing reduced by 70%. Marschall [70] found that releasing features as soon as they are done increases the developers' awareness of quality and their commitment, as they are able to see the link between their contribution and the quality of the end result.

Kettunen et al. [72] studied the differences in testing activities due to differences in process models. Organizations applying agile methods in general have to be more flexible in terms of testing practices in order to manage testing in parallel with development work. They found that early testing leads to more test execution time and a need for more predictable test resources.

Other process models than agile also have assessed rapid releases. In distributed software development, rapid releases were also found to be a practice that improves collaboration between distributed software development sites [51]. A case study from the UK [50] claims that the lean approach resulted in more rapid releases with shorter lead time that reduced both technical and market risks. Furthermore, the lean approach, which promotes continuous flow (i.e., continuous releases), indicates that workload peaks should be avoided for multiple reasons: defects are hidden and discovered late, the introduction of the defect has occurred a longer time ago (the person fixing it has difficulty to remember the whole context), waiting times are created, waiting times are reduced (SPI-LEAM), and stress-peaks are avoided [59]. Bell et al showed how defect prediction can be used even under a continuous release process with more frequent release dates [48].

6.4 Enablers - LR-RQ4

The enablers describe accelarating factors that facilitate the adoption of rapid releases, although they are not necessary preconditions. Several case studies [35, 36, 46] point out that frequent software releases are achieved through the parallel development of several releases. Hence, it is possible for a single release to have a long lead time on its own, but because development of different releases overlaps, the customer sees a frequent stream of new releases.

Olsson et al. [73] present a framework with key barriers for moving from traditional development to rapid releases in "innovation experiment systems". They

validate their approach with a case study in four companies. The given key barriers were the deployment of agile practices, automated testing, the involvement of product managers and pro-active customers. Other reports and companies point out that rapid releases require efficient build, test, and release infrastructure [35, 70].

A case study of a US Department of Defense (DoD) contractor [41], showed that DoD contractors also can have rapid releases. However, to enable rapid release all the way through to the production environment, one needs to accelerate the approval testing process of the software, which goes through 3 phases of external testing that can take up to 9 months. This problem was partially solved by starting the approval testing process simultaneously with the contractor testing process. The organization also planned to use more automated tests to further accelerate this process.

One area where release cycle time and software quality intersect is release planning, *i.e.*, "the selection and assignment of features to a sequence of consecutive product releases such that the most important technical, resource, budget and risk constraints are met" [74]. Either the release cycle is fixed, and features need to be selected to ensure a given quality level, or a feature set has been selected and the release cycle varies to ensure a given quality level. In principle, one could also vary the quality between releases, but in practice this is dangerous as users are expecting a certain quality level and deviating from that would reduce the user base permanently. A set of articles regarding release planning matched our search term "agile release", but when we looked into those articles the release length simply appeared as a variable in the release models without providing insights regarding rapid releases [75,76].

6.5 Negative Issues - LR-RQ5

There are a number of negative issues surrounding rapid releases. First, literature links rapid releases with problems of reliability and lack of testing coverage. Li et al. [44] point out that rapid releases and high reliability are conflicting aims. The authors propose a reliability model that finds an optimal compromise between having rapid releases and high reliability, and test the model with data from Apache and Gnome projects. Similarly, shorter release cycles make it impossible to test all possible configurations of a released product [8]. Furthermore, Petersen et al. [9] point out that in rapid releases the test cycles are often too short to conduct an extensive system test of quality attributes (e.g., performance), as these are more time-intensive.

Second, advocates of rapid releases and agile methods state that automated testing is the solution for reliability and testing problems. Although automated unit tests start becoming a standard practice in the industry, automated acceptance testing is more difficult to achieve, as only 27% of the respondents of a large practitioner survey used it (compared to 74% using unit tests) [68]. Similarly, case studies and surveys [77,78,79] show that automation is no silver bullet. For example, in a recent survey [79] only 6% of the respondents agreed that test automation can fully replace manual testing. An example of test automation problems comes from an Indian company that had implemented rapid releases [35]. The GUI-tests were created through manual testing, during which the testers recorded

their actions into an automated GUI-script. However, when using the recorded scripts afterwards, the company had problems recognizing when the automated tests failed, i.e., the test-oracle problem. Hence, they have testers to "watch the system manually as the automatic testing takes place". This was despite the fact that the company had used several different GUI testing tools. Thus, although automation can alleviate the problems of rapid releases it seems unlikely that it can completely solve the challenges related to reliability or test coverage.

Third, releasing too frequently not only decreases the time to run tests, but it might also make customers weary of yet another update to install [8,80]. This problem might be exacerbated by customers who have heavy formal acceptance processes for a new version or by customers that developed in-house applications on top of a software product that changes to a rapid release model. An example of the former is a DoD project where the formal acceptance phase could take 9 months [41], while an example of the latter is given by a Firefox corporate customer: The customer has thousands of internal apps that have to be tested and validated for each major Firefox release, thus, the adoption of rapid release model "is a kick in the stomach" [4]. For this reason, many projects do not automatically distribute each release to their customers. To cope with exactly this problem Firefox adopted an "Extended Support Release" (like Firefox 10.0, see **FF-RQ1**) that has a lifetime of 54 weeks instead of the standard six weeks [28]. Similarly, the Eclipse project uses 6-week release cycles, but the resulting milestone releases are only available to interested users and developers [66]. Naturally, clear communication about each release is necessary to make sure that only the intended user group deploys the new release [66,80].

Fourth, rapid releases may also increase technical debt as there is less time available for thinking things through. A case study of technical debt within a large US company (250 developers) [40] found that developers observe technical debt stemming from speed and lack of discipline in the development environment. Thus, the rapid releases may force software engineers and testers to cut some corners. Furthermore, a case study of 5 industrial projects (2 from Ericsson, the open source Linux kernel, FreeBSD and JBoss projects) [47] found that strict deadlines increase technical debt.

To control for technical debt in rapid releases, one must dedicate time to fix technical debt outside of the tight time frame of rapid releases [47]. Another way to avoid technical debt occurs in the Linux and FreeBSD projects, which do not set deadlines for tasks although they have a regular release schedule (3 month for Linux and 12 months for FreeBSD). Furthermore, low quality work is blocked by the pre-commit review phase [47], or alternatively, in a scrum project, one can have a separate backlog for such tasks. For example, Azham [49] suggests that having a separate backlog to deal with security issues can be a certain type of technical debt.

7 Discussion

Here, we compare the findings of the Firefox case study and the semi-systematic literature review, i.e., what do our findings in the case study add to existing work? Again, Figure 18 helps to obtain a high-level view of all our findings, both originating from the Firefox study and the literature review.

LR-RQ1) Where do RRs originate from?

In the literature review, we found that rapid releases are part of agile, open source, lean and internet speed development. On the other hand, our study on the transition to rapid releases of the Mozilla Firefox project is not tied to a new development paradigm such as agile or lean. Rather, in our study the move to rapid releases was market-driven, inspired by a competitor's strategy on frequent releases. Such changes are not necessarily driven by traditional software process improvement drivers like improving the software quality or decreasing the cost of the development process. Although those drivers played a role in the case of Firefox, the changes to the software development process correlated with faster crashes in the early RR releases [7], and created a need to hire more contractors for testing, as shown in **FF-RQ2**.

Firefox is an open source project and rapid releases are a practice of open source development. In the traditional release model, Firefox had the main version coming up roughly once a year and individual smaller releases every 26 days (median), see Figure 2. Thus, we cannot say that the traditional model had particularly infrequent releases since a release came out roughly once a month. However, when moving to RR the process became more organized with fixed, faster main releases coming out every six weeks and individual smaller releases coming out roughly once a week (median). Thus, the Firefox case can be summarized as moving from somewhat rapid, but irregular releases to predictable, rapid releases because of market-driven reasons.

LR-RQ2) What is the prevalence of RRs?

The literature shows how rapid releases are adopted in several domains such as banking, internet software, telecom, and even in projects of the US Department of Defense (DoD). The Firefox case study does not add any new domain as internet-related products have been previously investigated in the context of rapid releases, e.g., [7,63]. Additionally, the literature shows that 23% of practitioners perform continuous releases, while 83% of practitioners release at least once every three months. This case study showed how Firefox release cycles went down from 26 days to 7 days.

LR-RQ3) What are the benefits of RRs?

In the literature, several benefits of RR were mentioned: rapid feedback, improved quality focus of the developers and testers, easier monitoring of progress and quality, customer satisfaction, shorter time-to-market, and increased efficiency due to increased time-pressure. Our study brings an important addition to this list, which is the smaller scope of testing that allows targeted, in-depth testing. Thus, although there is less time for testing, testing is more manageable due to its smaller scope. We also found that in RR releases, testing is more continuous, whereas in TR the variation in test executions over time is higher.

LR-RQ4) What are the enablers of RRs?

From literature, we found that rapid releases were enabled through parallel development, with tools enabling easy automatic deployment and testing, and with proactive customers and product managers. In the case of Firefox, frequent releases

are created with a short lead time, but they also work on releases simultaneously, as we saw in Figure 1. Additionally, we found that dedicated and paid testing resources enabled rapid release testing. Additionally, we also found evidence both from qualitative interviews and statistical data analysis that testing has become more focused. Thus, rather than trying to cover everything, only the high risk areas receive significant test effort. Overall, Mozilla's strategies for testing RR releases have been succesful as Khomh et al. [7] found only a small decrease in quality when investigating the move to rapid release in the Firefox project.

LR-RQ5) What are the problems of RRs?

In literature, the problems related to rapid releases were increased technical debt, conflicts with high reliability and high test coverage, problems with the customer adoption of releases, and increased time-pressure that can lead to staff burnout. Regarding technical debt, we found no evidence of cutting corners. Our findings partially support these findings, as we found that Firefox had reduced the legacy support in their testing and that a reduction in the testing community had reduced the number of environments that were being tested. On the other hand, the Firefox QA engineer also stated that in-depth testing of a few features was one of the greatest benefits of rapid releases. The decline in quality or reliability in the Firefox case has indeed been small, since rapid releases caused no change in the post-release defect counts or number of crashes experienced, except for crashes appearing faster for the users of rapid releases and fewer defects being fixed [7]. Problems of customer adoption were solved in the Firefox case by adoption of an "Extended Support Release" (like Firefox 10.0, see **FF-RQ1**) with longer release lifetime.

In addition to literature, we showed that rapid releases had led to a decrease in the test-suite (test case set) and tester diversity between releases, i.e., there is a larger overlap of testers participating and test cases being executed between subsequent releases. Diversity in testing is important, but we are unaware of prior work on test-suite and tester diversity, although diversity [81] and similarity [82] have been studied at the level of individual test cases. Also, the number of testers participating in the release has decreased, as the community participation has decreased. We also found that rapid release may increase time-pressure as testing becomes more deadline-oriented, as shown in **FF-RQ6**. Time-pressure has been linked to burn-out in software development organizations [83] and technical debt, but on the positive side, experiments on time-pressure have shown an increase in individual efficiency for software testing [84] and other types of tasks, e.g., accounting [85].

7.1 Limitations

7.1.1 Firefox study

Every empirical study has limitations. In this section we structure our analysis of the limitations according to the framework by Runeson et al. [86]: construct validity, internal validity, external validity and reliability.

Construct validity concerns the match between our research questions and what we observe or measure. In our study, the selection of metrics and analysis methods

Table 12: Controlled and non-controlled confounding factors.

Controlled	Uncontrolled
Project evolution, Release length	Competing projects, Other type of testing (e.g., dog-fooding)

is critical. To enable comparison between releases of varying length we normalized all metrics for project duration. Our data was not normally distributed so all our statistical analysis including effect sizes are computed with non-parametric methods. We mapped test executions to individual releases based on timestamp and the main release label. The main release label allowed us to correctly differentiate between simultaneous testing on different channels, e.g., 5.0 beta and 6.0 alpha could be tested simultaneously but labels of 5.0 and 6.0 in addition to the timestamps allowed us to correctly identify the release. Since it is a case study, another threat to construct validity is our interpretation of the context of the project. To control for this, we triangulated our findings with a Mozilla engineer.

Internal validity concerns causal relationships. 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. Again, there is a difference between a case study and a controlled experiment and we cannot interpret our statistical findings in isolation. There could be hidden factors that actually cause these observed differences, such as the competing projects in **FF-RQ2** or dog-fooding in **FF-RQ6**. The confounding effect of project evolution and release length was thoroughly studied in Section 5. Table 12 summarized the identified confounding factors. Yet, more studies are needed before affirmative conclusions on the effects of a release model on testing effort can be made.

External validity concerns the ability to generalize our findings. Although we study over 200 Firefox releases, our study only considers one open source system. Moreover, the Litmus database represents only one part of the Firefox testing process that 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.

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. Although statistical generalization between contexts cannot be made based on the results from this study, analytical generalization is supported through the context description in the paper. Our study highlights possible effects on testing due to a transition from TR to RR releases.

Reliability regards the reproducibility of our results. All observations are interpreted by researchers and thus also filtered through their perceptions, knowledge and experiences. In our case, five researchers with different backgrounds have been involved in the interpretation of the observations, which supports the reliability. We also provide transparency in the report of the study design, data collection and analysis.

7.1.2 Literature Review

In many abstracts, rapid releases were mentioned, yet no actual empirical result of rapid releases was given in the paper. We suspect that also the dual situation can be true, i.e., studies that do not advertise results regarding rapid releases in their abstract. For this reason, all authors also performed other types of literature search, but with a less systematic approach (i.e., papers that we could recall, papers found by full text search using google scholar, or papers studying agile adoption that did not explicitly mention our search terms in the abstract). Despite this limitation, we think that the literature review's results offer the most comprehensive review of evidence of rapid releases to date. However, future reviews of rapid releases should systematically search for agile and lean adoption papers, as such papers can contain data about rapid releases.

8 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, are performed more continuously 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 as has test suite diversity. Third, comparison to Khomh et al.'s work [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.

Additionally, this paper presents a semi-systematic literature review showing that rapid releases are a prevalent industrial practice that affects several domains of software engineering and is part of several software development methodologies like agile, lean and open source. The benefits of rapid releases according to literature include shorter time to market, rapid feedback and an increased quality focus of development staff. Furthermore, moving to rapid releases is enabled by parallel development, tools for automatic deployment and testing, and the involvement of customers. Finally, the negative aspects of rapid releases are increased technical debt, the customers' (un)willingness to update, time pressure, and the conflicting goals of releasing rapidly and achieving high reliability and test coverage. Future work should focus on empirical studies of these factors that complement the existing qualitative observations and perceptions of rapid releases.

Acknowledgment

We would like to thank the Mozilla QA engineer who provided feedback to our findings. His responses are accounts of personal experience and opinion, and are in no means whatsoever an official statement from Mozilla. This work has been partially supported by EU FP7 Grant 318082 (U-QASAR, http://www.uqasar.eu/), ELLIIT (the StrategicArea for ICT research, funded by the Swedish Government, http://www.liu.se/elliit), N4S (research program managed by DIGILE and funded by TEKES, http://www.n4s.fi/), and NSERC (Natural Sciences and Engineering Research Council of Canada).

References

1. HP: Shorten release cycles by bringing developers to application lifecycle management. HP Applications Handbook, Retrieved on Febuary 08, 2012 (2012)
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` (2011)
3. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley (2004)
4. Shankland, S.: Rapid-release firefox meets corporate backlash. http://cnet.co/ktBsUU (2011)
5. Kaply, M.: Why do companies stay on old technology? Retrieved on January 12, 2012 (2012)
6. Baysal, O., Davis, I., Godfrey, M.W.: A tale of two browsers. In: Proc. of the 8th Working Conf. on Mining Software Repositories (MSR). (2011) 238–241
7. Khomh, F., Dhaliwal, T., Zou, Y., Adams, B.: Do faster releases improve software quality? an empirical case study of mozilla firefox. In: MSR. (2012) 179–188
8. Porter, A., Yilmaz, C., Memon, A.M., Krishna, A.S., Schmidt, D.C., Gokhale, A.: Techniques and processes for improving the quality and performance of open-source software. Software Process: Improvement and Practice **11** (2006) 163–176
9. Petersen, K., Wohlin, C.: The effect of moving from a plan-driven to an incremental software development approach with agile practices. Empirical Softw. Engg. **15** (2010) 654–693
10. Li, J., Moe, N.B., Dybå, T.: 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) 13:1–13:10
11. Mantyla, M., Khomh, F., Adams, B., Engstrom, E., Petersen, K.: On rapid releases and software testing. In: Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM). (2013) 20–29
12. Ltd., R.S.: Web browsers (global marketshare). `http://bit.ly/81klgi` (2013)
13. Shankland, S.: Google ethos speeds up chrome release cycle. http://cnet.co/wlS24U (2010)
14. Sicore, D.: New channels for firefox rapid releases. `http://bit.ly/hc1zmY` (2011)
15. Paul, R.: Mozilla outlines 16-week firefox development cycle. `http://bit.ly/fLHEfo` (2011)
16. Mozilla: Litmus wiki. `http://mzl.la/evJmTW` (2013)
17. Mozilla: Moztrap wiki. `http://bit.ly/XBGMfu` (2013)
18. Wikipedia: Firefox release history. `http://bit.ly/Ngvfln` (2013)
19. Mozilla: Mozilla source code mercurial repositories. (2013)
20. Cohen, J.: Statistical Power Analysis for the Behavioral Sciences. 2nd edn. Academic Press (1988)
21. Rogmann, J.J.: orddom: Ordinal dominance statistics. `http://bit.ly/Y0K0eo` (2013)
22. Byrt, T., Bishop, J., Carlin, J.B.: Bias, prevalence and kappa. Journal of clinical epidemiology **46** (1993) 423–429
23. Cicchetti, D.V., Feinstein, A.R.: High agreement but low kappa: Ii. resolving the paradoxes. Journal of clinical epidemiology **43** (1990) 551–558
24. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. biometrics (1977) 159–174

25. Carney, P.A., Sickles, E.A., Monsees, B.S., Bassett, L.W., Brenner, R.J., Feig, S.A., Smith, R.A., Rosenberg, R.D., Bogart, T.A., Browning, S., et al.: Identifying minimally acceptable interpretive performance criteria for screening mammography 1. Radiology **255** (2010) 354–361
26. Pérez-Castillo, R., Sánchez-González, L., Piattini, M., García, F., Garcia-Rodriguez de Guzman, I.: Obtaining thresholds for the effectiveness of business process mining. In: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, IEEE (2011) 453–462
27. Hollander, M., Wolfe, D.A.: Nonparametric Statistical Methods. 2nd edn. John Wiley and Sons, inc. (1999)
28. Wikipedia: Extended support release. `http://bit.ly/ZlgqoM#Extended_Support_Release` (2013)
29. Paul, R.: Firefox extended support will mitigate rapid release challenges. `http://ars.to/M2TbFQ` (2012)
30. Wikipedia: Locale. `http://bit.ly/2iJLwB` (2013)
31. Raymond, E.S.: The Cathedral and the Bazaar. 1st edn. O'Reilly & Associates, Inc., Sebastopol, CA, USA (1999)
32. Kim, S.: ppcor: Partial and semi-partial (part) correlation. `http://bit.ly/XkWuyn` (2012)
33. Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33** (2004) 2004
34. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. 1st edn. Addison-Wesley Professional (2010)
35. Agarwal, P.: Continuous scrum: Agile management of saas products. (2011) 51–60
36. Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J.: High-speed software development practices: What works, what doesn't. IT Professional **8** (2006) 29–36
37. Baysal, O., Kononenko, O., Holmes, R., Godfrey, M.: The secret life of patches: A firefox case study. (2012) 447–455
38. Begel, A., Nagappan, N.: Usage and perceptions of agile software development in an industrial context: An exploratory study. (2007) 255–264
39. Boshuizen, C., Marshall, W., Bridges, C., Kenyon, S., Klupar, P.: Learning to follow: Embracing commercial technologies and open source for space missions. Volume 10. (2011) 8097–8101
40. Codabux, Z., Williams, B.: Managing technical debt: An industrial case study. (2013) 8–15
41. Cohan, S.: Successful integration of agile development techniques within disa. (2007) 255–260
42. Kühner, G., Bluhm, T., Heimann, P., Hennig, C., Kroiss, H., Krom, J., Laqua, H., Lewerentz, M., Maier, J., Schacht, J., Spring, A., Werner, A., Zilker, M.: Progress on standardization and automation in software development on w7x. Fusion Engineering and Design **87** (2012) 2232–2237
43. Lavoie, T., Merlo, E.: How much really changes? a case study of firefox version evolution using a clone detector. (2013) 83–89
44. Li, X., Li, Y., Xie, M., Ng, S.: Reliability analysis and optimal version-updating for open source software. Information and Software Technology **53** (2011) 929–936
45. Maalej, W.: Task-first or context-first? tool integration revisited. (2009) 344–355
46. Sundmark, D., Petersen, K., Larsson, S.: An exploratory case study of testing in an automotive electrical system release process. (2011) 166–175
47. Torkar, R., Minoves, P., Garrigós, J.: Adopting free/libre/open source software practices, techniques and methods for industrial use. Journal of the Association of Information Systems **12** (2011) 88–122
48. Bell, R., Ostrand, T., Weyuker, E.: Looking for bugs in all the right places. Volume 2006. (2006) 61–71
49. Azham, Z., Ghani, I., Ithnin, N.: Security backlog in scrum security practices. (2011) 414–417
50. Middleton, P., Joyce, D.: Lean software management: Bbc worldwide case study. IEEE Transactions on Engineering Management **59** (2012) 20–32
51. Paasivaara, M., Lassenius, C.: Collaboration practices in global inter-organizational software development projects. Software Process Improvement and Practice **8** (2003) 183–199
52. Patel, C., Lycett, M., Macredie, R., De Cesare, S.: Perceptions of agility and collaboration in software development practice. Volume 1. (2006) 10c

53. Gundebahar, M., Kus Khalilov, M.: A hybrid deployment model for financial systems with service oriented architecture (soa): Running from client via branch server. (2013) 365–370
54. Iwai, A., Aoyama, M.: Automotive cloud service systems based on service-oriented architecture and its evaluation. (2011) 638–645
55. Jones, G., Leung, V.: Visual surveillance: A systems engineering approach for rapid development. Number 2005-11033 (2005) 161–166
56. Olsson, H., Alahyari, H., Bosch, J.: Climbing the "stairway to heaven" - a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. (2012) 392–399
57. Schwaber, K.: Scrum development process. In: Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA. (1995) 117–134
58. Otte, T., Moreton, R., Knoell, H.D.: Applied quality assurance methods under the open source development model. In: Proc. of the 32nd Annual IEEE Intl. Computer Software and Applications Conf. (COMPSAC). (2008) 1247–1252
59. Middleton, P.: Lean software development two case studies. Software Quality Journal **9** (2001) 241–252
60. Poppendieck, M.: Lean software development. In: Companion to the proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society (2007) 165–166
61. Reinertsen, D.G.: The principles of product development flow: second generation lean product development. Celeritas Redondo Beach, Canada: (2009)
62. Oza, N., Ebert, C., Abrahamsson, P.: Lean software development. IEEE Software **29** (2012) 0022–25
63. Cusumano, M.A., Yoffie, D.B.: Competing on internet time: Lessons from netscape and its battle with microsoft. Simon and Schuster (1999)
64. Brown, A.W.: 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). Volume 77. (2011) 266–281
65. Jenkins, J.: Velocity culture (the unmet challenge in ops). Presentation at O'Reilly Velocity Conference (2011)
66. Gamma, E.: Agile, open source, distributed, and on-time – inside the eclipse development process. Keynote at the 27th Intl. Conf. on Software Engineering (ICSE) (2005)
67. Kong, S., Kendall, J.E., Kendall, K.E.: The challenge of improving software quality: Developers' beliefs about the contribution of agile practices. In: Proc. of the Americas Conf. on Information Systems (AMCIS). (2009) 12p.
68. VersionOne: 7th annual state of agile survey. http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf (2012)
69. Hodgetts, P., Phillips, D.: 30. In: eXtreme Adoption eXperiences of a B2B Start Up. Addison-Wesley Longman Publishing Co., Inc. (2002) Extreme Programming Perspectives.
70. Marschall, M.: Transforming a six month release cycle to continuous flow. In: Proc. of the conf. on AGILE. (2007) 395–400
71. Petersen, K., Wohlin, C.: A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. J. Syst. Softw. **82** (2009) 1479–1490
72. Kettunen, V., Kasurinen, J., Taipale, O., Smolander, K.: A study on agility and testing processes in software organizations. In: Proc. of the 19th Intl. Symp. on Software Testing and Analysis (ISSTA). (2010) 231–240
73. Olsson, H., Bosch, J., Alahyari, H.: Towards r&d as innovation experiment systems: A framework for moving beyond agile software development. (2013) 798–805
74. Ruhe, G., Saliu, M.O.: The art and science of software release planning. IEEE Softw. **22** (2005) 47–53
75. Heikkilä, V., Rautiainen, K., Jansen, S.: A revelatory case study on scaling agile release planning. (2010) 289–296
76. Szőke, Á.: Conceptual scheduling model and optimized release scheduling for agile environments. Information and Software Technology **53** (2011) 574–591
77. Berner, S., Weber, R., Keller, R.K.: Observations and lessons learned from automated testing. In: Proceedings of the 27th international conference on Software engineering, ACM (2005) 571–579

78. Martin, D., Rooksby, J., Rouncefield, M., Sommerville, I.: 'good'organisational reasons for'bad'software testing: An ethnographic study of testing in a small software company. In: Software Engineering, 2007. ICSE 2007. 29th International Conference on, IEEE (2007) 602–611
79. Rafi, D.M., Moses, K.R.K., Petersen, K., Mäntylä, M.V.: Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: Automation of Software Test (AST), 2012 7th International Workshop on, IEEE (2012) 36–42
80. Jansen, S., Brinkkemper, S.: Ten misconceptions about product software release management explained using update cost/value functions. In: Proc. of the Intl. Workshop on Software Product Management. (2006) 44–50
81. Hemmati, H., Arcuri, A., Briand, L.: Achieving scalable model-based testing through test case diversity. ACM Transactions on Software Engineering and Methodology (TOSEM) **22** (2013) 6
82. Greiler, M., van Deursen, A., Zaidman, A.: Measuring test case similarity to support test suite understanding. In: Objects, Models, Components, Patterns. Springer (2012) 91–107
83. Sonnentag, S., Brodbeck, F.C., Heinbokel, T., Stolte, W.: Stressor-burnout relationship in software development teams. Journal of occupational and organizational psychology **67** (1994) 327–341
84. Mäntylä, M.V., Itkonen, J.: More testers âĂŞ the effect of crowd size and time restriction in software testing. Information and Software Technology **55** (2013) 986 – 1003
85. McDaniel, L.S.: The effects of time pressure and audit program structure on audit performance. Journal of Accounting Research **28** (1990) 267–285
86. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14** (2009) 131–164