

# Marco Castelluccio

Tutor: Carlo Sansone – co-Tutor: Luisa Verdoliva

XXXI Cycle - III year presentation

Improving software engineering  
processes using machine learning  
and data mining techniques

# Background

- MSc with honour in Computer Engineering at University of Naples Federico II
- Senior Software Engineer at Mozilla



# Collaborations

- SWAT (SoftWare Analytics and Technologies) team at Ecole Polytechnique de Montréal
- ZEST (Zurich Empirical Software engineering Team) team at University of Zurich

# Credits

Year	Modules	Seminars	Research	Tot.
1	21 (20)	7 (5)	35 (35)	63 (60)
2	16 (9)	12 (6)	45 (42)	73 (60)
3	3 (0)	10 (5)	51 (55)	64 (60)
<b>Tot.</b>	<b>40 (30-70)</b>	<b>29 (10-30)</b>	<b>131 (80-140)</b>	<b>180 (180)</b>

# Firefox

# Firefox

## Misc Stats

- Releases every 6 to 8 weeks
- One of the biggest and most complex software, with a little bit of legacy code and tech debt (Netscape was opensourced 20 years ago!)
- 399'221 commits from 5'356 unique contributors, around 18'000'000 lines of code
- Around 60'000 commits last year, from 1'267 unique contributors

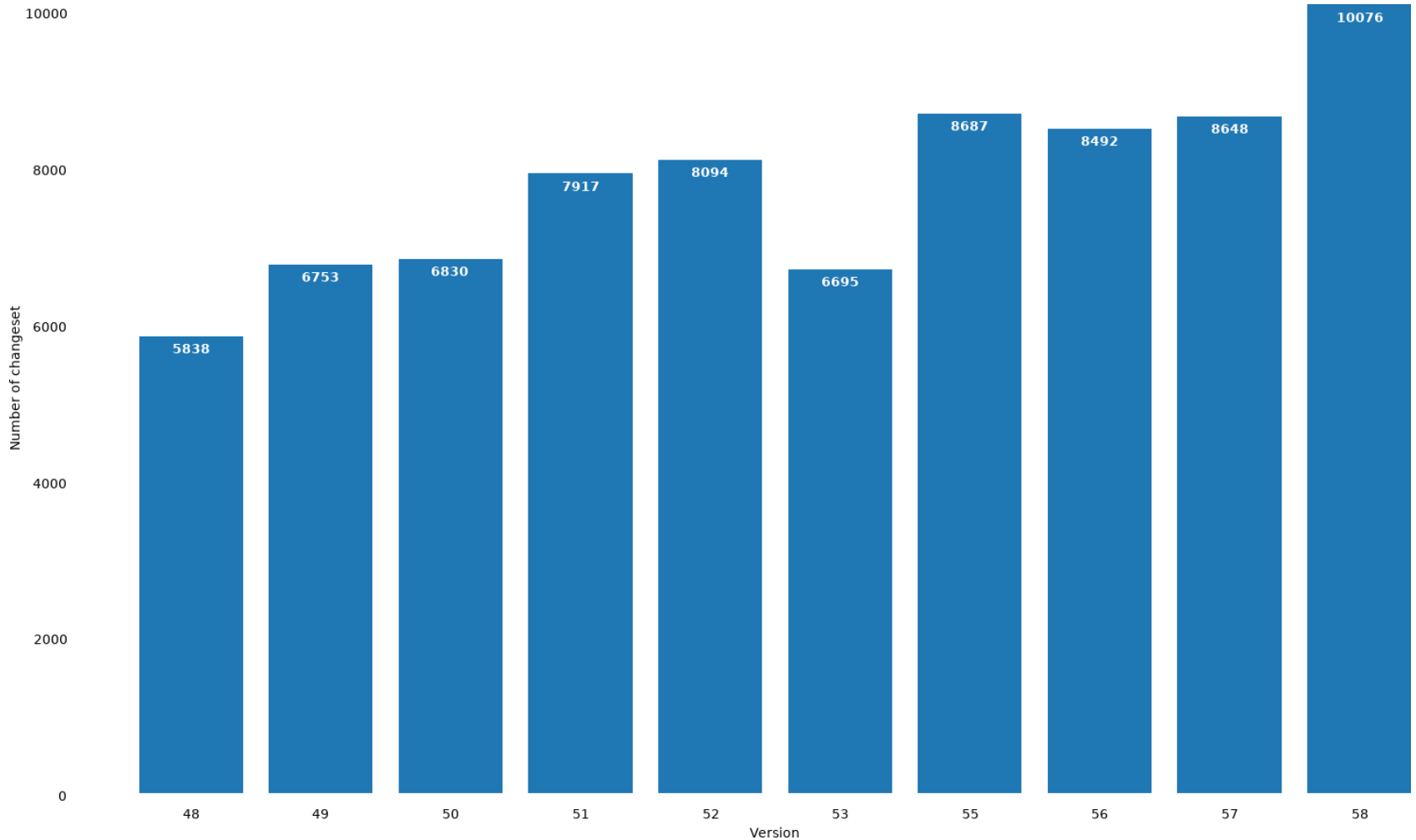
# Firefox

## Languages

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines
C++	5,819,797	1,196,317	17.1%	1,092,288	8,108,402
JavaScript	3,894,509	1,292,625	24.9%	870,402	6,057,536
HTML	2,513,253	124,985	4.7%	309,884	2,948,122
C	2,397,973	639,359	21.1%	400,372	3,437,704
Rust	801,636	154,096	16.1%	94,367	1,050,099
XML	720,947	16,434	2.2%	41,484	778,865
Python	602,906	174,837	22.5%	157,613	935,356
Java	327,613	122,062	27.1%	67,216	516,891
Assembly	226,062	24,811	9.9%	30,734	281,607
CSS	225,507	14,186	5.9%	32,804	272,497
Autoconf	104,687	1,843	1.7%	14,112	120,642
shell script	87,896	16,985	16.2%	13,209	118,090
Objective-C	57,098	8,665	13.2%	11,818	77,581
Make	49,840	14,709	22.8%	13,116	77,665
OpenGL Shading	32,369	34,703	51.7%	10,356	77,428
Perl	17,019	3,358	16.5%	3,804	24,181
NSIS	10,449	2,959	22.1%	2,195	15,603
CMake	7,301	2,005	21.5%	1,550	10,856
TeX/LaTeX	6,097	3,230	34.6%	752	10,079
DOS batch script	3,294	163	4.7%	524	3,981
Automake	3,212	222	6.5%	298	3,732

# Firefox

## Patches landed in last releases





# Firefox

## CI

- Tests executed on every commit, under several configurations and on different platforms
- The average execution time of all tests is 1'506 hours
- In November 2017, 8'319'189 tasks, 299,8 machine years, 927'333 machines

# Firefox CI

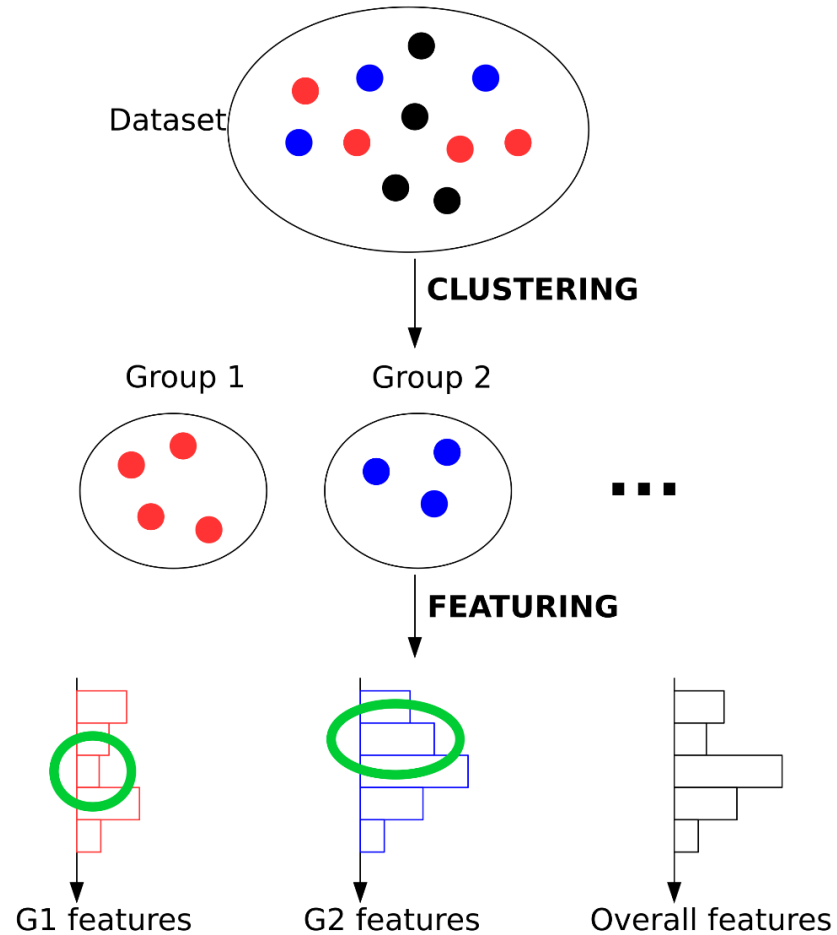
The screenshot shows the treeherder CI interface. At the top, there's a navigation bar with 'treeherder' and various filters like 'Infra', 'Repos', 'Tiers', 'Filters', and 'Login/Register'. Below that, a status bar indicates '6 unclassified' jobs and 'Filter platforms & jobs'. The main content area shows a list of build jobs for 'mozilla-central' on 'try' and 'mozilla-inbound' platforms. The jobs are categorized by platform: Linux opt, Linux pgo, Linux debug, Linux Stylo Disabled opt, Linux Stylo Disabled debug, and Linux x64 opt. Each job entry includes a list of test results, such as 'B Cpp GTest Mn N Fxfr-l-e10s(en-US) Fxfr-r-e10s[...]' and 'M(a11y c1 c2 c3) M-e10s(1 2 3 4 5)'. The interface also shows 'Active Filters' and a 'revision: c1c444858b32'.

# Automatic understanding groups of crashes for finding correlations

## ESEC/FSE 2017

# Automatic understanding groups of crashes for finding correlations

## Crash-analysis workflow



# Automatic understanding groups of crashes for finding correlations

## Socorro

- Socorro is the crash-reporting system deployed at Mozilla
- It handles around 200'000 reports per day from multiple Firefox builds
- It uses a simple clustering algorithm to group crash reports (top method of the stack trace), generating thousands of clusters (however, most of them are really small, with the first 200 clusters containing 55% of reports)

# Automatic understanding groups of crashes for finding correlations

## Socorro – Crash report fields

Name	Description
Platform	The name of the Operating System.
Platform Version	The detailed version of the Operating System (e.g. <i>uname -a</i> on Linux).
Addons	A list of the addons, with their version, installed in the Firefox profile.
Modules	A list of the modules (DLL files on Windows, SO files on Linux, dylib files on Mac), with their version, loaded in the application's process.
User Comment	A (usually brief) comment left by the user at the time of crashing.
CPU Info	Detailed information (vendor, family, model, stepping, number of cores) about the CPU of the user.
Adapter Vendor ID	The vendor of the graphics card on the user's machine. There are other related attributes such as Adapter Device ID, Adapter Driver Version, etc.
Safe Mode	A boolean variable that indicates whether Firefox was running in safe mode.
User Agent Locale	The language of the user.
...	...

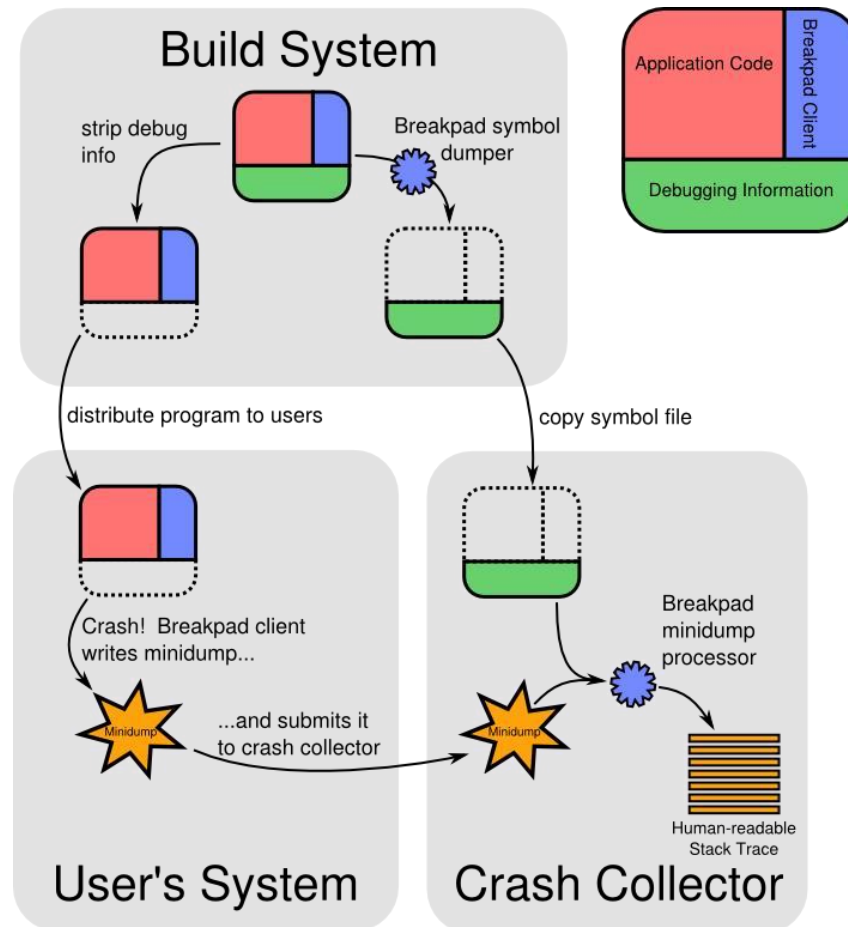
# Automatic understanding groups of crashes for finding correlations

## Socorro – Crash report stack trace

Frame	Module	Signature
0	xul.dll	mozilla::storage::Service::getSingleton()
1	xul.dll	mozilla::storage::ServiceConstructor
2	xul.dll	nsComponentManagerImpl::CreateInstanceByContractID(char const*, nsISupports*, nsID const&, void**)
3	xul.dll	nsComponentManagerImpl::GetServiceByContractID(char const*, nsID const&, void**)
4	xul.dll	nsCOMPTr_base::assign_from_gs_contractid(nsGetServiceByContractID, nsID const&)
5	xul.dll	nsCOMPTr<mozIStorageService>::nsCOMPTr<mozIStorageService>(nsGetServiceByContractID)
6	xul.dll	nsPermissionManager::OpenDatabase(nsIFile*)
7	xul.dll	nsPermissionManager::InitDB(bool)
8	xul.dll	nsPermissionManager::Init()
9	xul.dll	nsPermissionManager::GetXPCOMSingleton()
10	xul.dll	nsIPermissionManagerConstructor
11	xul.dll	nsComponentManagerImpl::CreateInstanceByContractID(char const*, nsISupports*, nsID const&, void**)
12	xul.dll	nsComponentManagerImpl::GetServiceByContractID(char const*, nsID const&, void**)
13	xul.dll	nsCOMPTr_base::assign_from_gs_contractid(nsGetServiceByContractID, nsID const&)
14	xul.dll	nsCOMPTr<nsIPermissionManager>::nsCOMPTr<nsIPermissionManager>(nsGetServiceByContractID)
15	xul.dll	mozilla::services::GetPermissionManager()
16	xul.dll	mozilla::dom::NotificationTelemetryService::RecordPermissions()
17	xul.dll	NotificationTelemetryServiceConstructor
18	xul.dll	nsComponentManagerImpl::CreateInstanceByContractID(char const*, nsISupports*, nsID const&, void**)
19	xul.dll	nsComponentManagerImpl::GetServiceByContractID(char const*, nsID const&, void**)
20	xul.dll	nsCOMPTr_base::assign_from_gs_contractid(nsGetServiceByContractID, nsID const&)
21	xul.dll	nsCOMPTr<nsISupports>::nsCOMPTr<nsISupports>(nsGetServiceByContractID)
22	xul.dll	NS_CreateServicesFromCategory(char const*, nsISupports*, char const*, char16_t const*)
23	xul.dll	nsXREDirProvider::DoStartup()
24	xul.dll	XREMain::XRE_mainRun()
25	xul.dll	XREMain::XRE_main(int, char** const, nsXREAppData const*)
26	xul.dll	XRE_main
27	firefox.exe	do_main
28	firefox.exe	wmain
29	firefox.exe	__srt_common_main_seh
30	kernel32.dll	BaseThreadInitThunk
31	ntdll.dll	__RtlUserThreadStart
32	ntdll.dll	_RtlUserThreadStart

# Automatic understanding groups of crashes for finding correlations

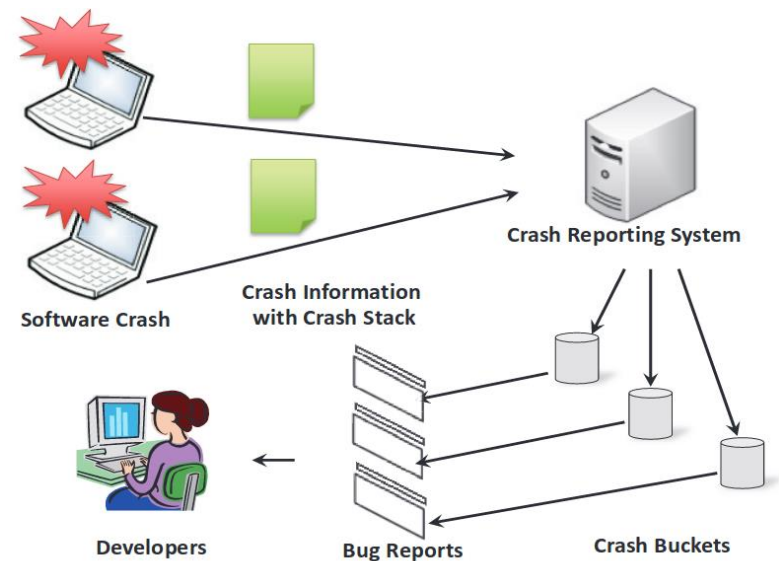
## Socorro – Architecture overview





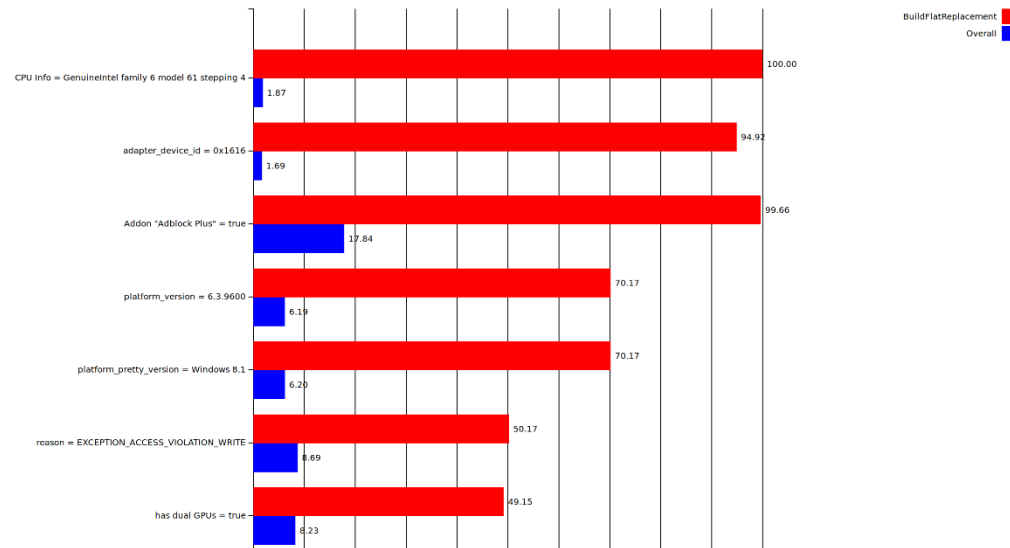
# Automatic understanding groups of crashes for finding correlations

- Many studies focused on improving the bucketing of crash reports. My focus has been on how to automatically describe the buckets' properties in the most interesting way for developers.



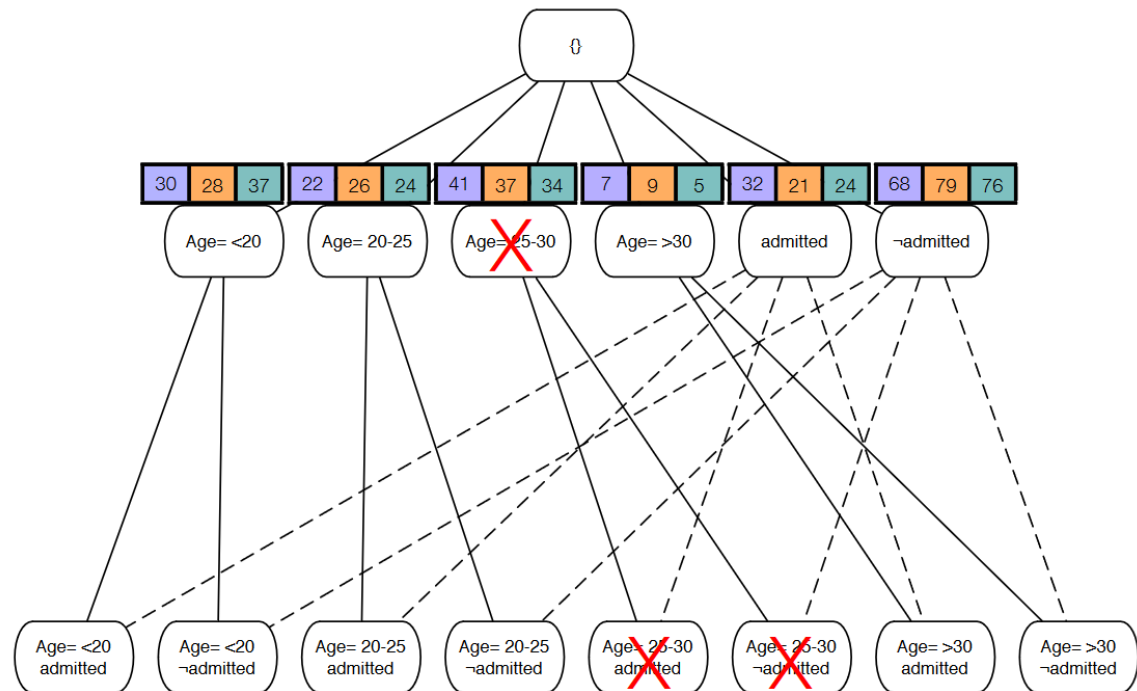
# Automatic understanding groups of crashes for finding correlations

- Understanding what makes a crash group meaningfully different than other groups is very often useful for debugging (sometimes even enough for fixing the crash, e.g. by blocklisting a certain gfx card).



# Automatic understanding groups of crashes for finding correlations

- The algorithm is based on the STUCCO data mining algorithm.



# Automatic understanding groups of crashes for finding correlations

## Results

- The tool implementing the algorithm was deployed on Socorro
- Evaluation on a set of bugs where developers used the tool in production

# Automatic understanding groups of crashes for finding correlations

## Deployment on Socorro

Signature report for *mozilla::dom::quota::QuotaManager::ShutdownObserver::Observe*

Showing results from 7 days ago to a minute ago.

Summary Aggregations Reports Graphs Bugzilla Comments **Correlations** Build Graph

Product:

Channel:

### Correlations for Firefox Release

(99.19% in signature vs 01.09% overall) address = 0x7c  
(100.0% in signature vs 05.84% overall) shutdown\_progress = profile-before-change  
(95.46% in signature vs 04.72% overall) Module "EFACli.dll" = true  
(93.78% in signature vs 03.90% overall) Module "ccvtrst.dll" = true  
(93.63% in signature vs 03.90% overall) Module "ccipc.dll" = true  
(95.49% in signature vs 06.29% overall) Module "IPSEng32.dll" = true  
(92.87% in signature vs 03.84% overall) Module "cclib.dll" = true  
(95.07% in signature vs 06.53% overall) Module "msvcpl10.dll" = true  
(95.07% in signature vs 06.74% overall) Module "msvcr110.dll" = true  
(100.0% in signature vs 35.31% overall) reason = EXCEPTION\_ACCESS\_VIOLATION\_READ

# Automatic understanding groups of crashes for finding correlations

## Results

Type	Number of bugs
Very useful – results that directly helped fixing the bug.	19
Compatible – results that were compatible with the resolution of the bug, but were not useful for fixing the bug.	19
Misleading – results not compatible with the resolution of the bug.	3

# Automatic understanding groups of crashes for finding correlations

## Results – How clustering affects results

- **Clusters containing unrelated crashes:** more difficult to find defining properties. E.g. crashes related to the JavaScript JIT compiler are often wrongly clustered together. The correlation tool is only able to tell that the group is related to the JIT.
- **Related crashes split in multiple clusters:** more likely to find spurious correlations. E.g. there was a crash, later diagnosed to be due to concurrency issues, happening in different functions according to CPU brand/graphics card. This caused the clusters to be highly but spuriously correlated with those properties.

# Automatic understanding groups of crashes for finding correlations

## Results – Interesting examples

- **AMD CPU bug:** A crash group was found to be correlated with a particular family of AMD CPUs. The particular family of AMD CPUs in the crash group is affected by a hardware bug, and developers were able to find a workaround for it
- **Antivirus-related crash:** A crash group was found to be correlated with a version of an addon of an antivirus suite. In cases like this, the tool allows practitioners to act quickly and simply block the addons (or modules) that cause problems, while contacting the vendors to solve the problem in the long term
- **Crash without AdBlock:** A crash group was more common to users without ad-blocking addons, often happening with a very famous Flash game. We believe the crash was caused by some advertising network serving particular advertisement that would cause the browser to crash.



# Automatic understanding groups of crashes for finding correlations

## Conclusions

- Analyzing crash groups in an automated manner can help:
  - Removing manual analysis burden from developers;
  - Finding properties that would have been really difficult to find with manual analysis;
  - Giving clues in the characterization of crashes.
- The results of the correlations tool could be useful in the future for:
  - Improving the clustering algorithm;
  - Suggesting hardware/software configurations likely to fail to QA and volunteers.

# Empirical study of the uplift process at Mozilla

ICSME 2017

+

EMSE

# Empirical study of the uplift process at Mozilla

What is an uplift?

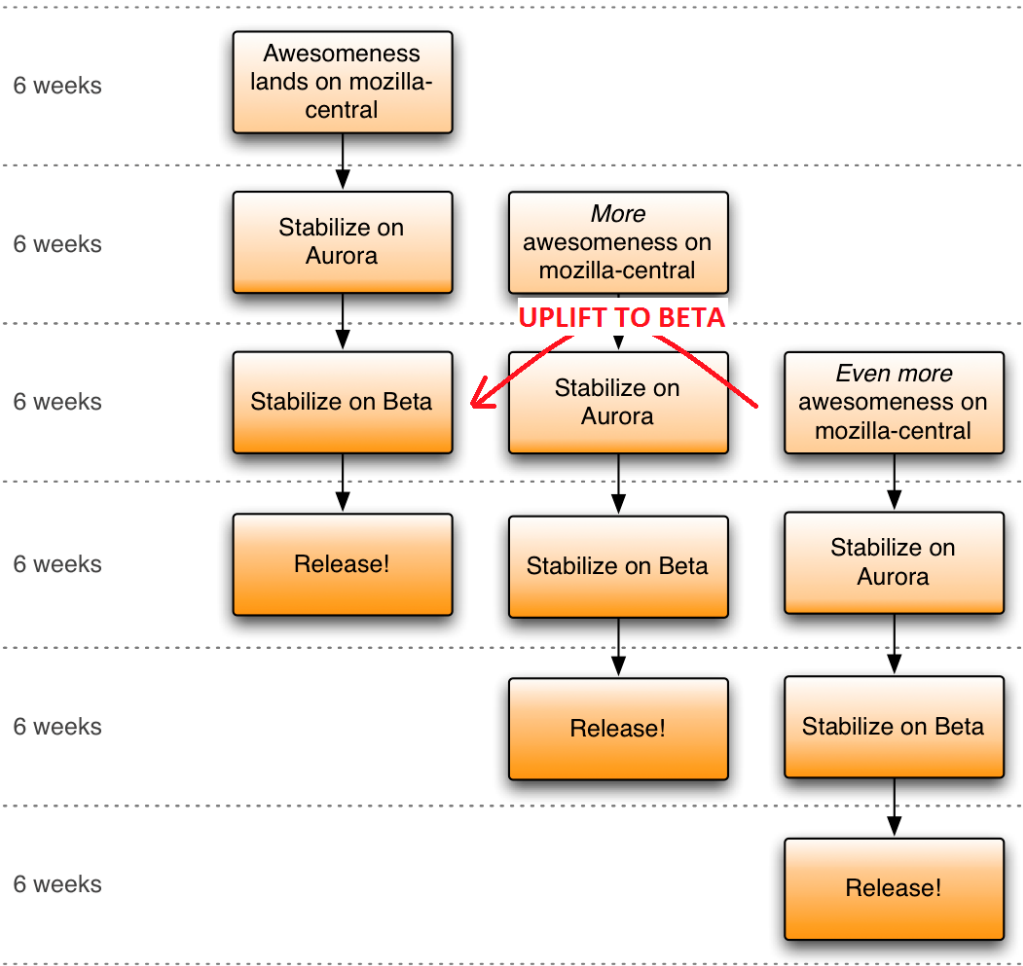
- In rapid-release development processes, high-value patches (fixing high volume crashes, introducing important features, fixing widespread regressions, etc.) are often promoted directly from the development branch (Nightly) to a stabilization channel, skipping one or more channels.
- Developers are requesting the uplifts.
- Release managers are in charge of the uplift process.

# Empirical study of the uplift process at Mozilla

- Collaboration with the École Polytechnique de Montréal.
- The aim is to understand the properties of uplift vs normal changes; understand which uplifts introduced bugs and why; with the ultimate goal of building a model to predict the riskiness of an uplift.

# Empirical study of the uplift process at Mozilla

## What is an uplift?



# Empirical study of the uplift process at Mozilla

## Methodology

- Focused on the timespan between September 2014 to August 2016, as it was a steady period, ignoring Pocket-related uplifts (since they were a one-time event). Total of ~40000 bugs, ~7000 uplifts.

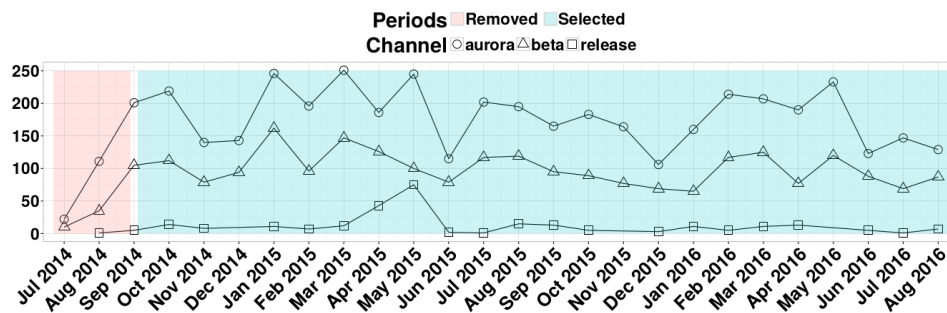


Figure 1: Number of uplifts during each month from July 2014 to August 2016. Periods with low number of uplifts or not covering all the three channels are removed.

# Empirical study of the uplift process at Mozilla

## Methodology

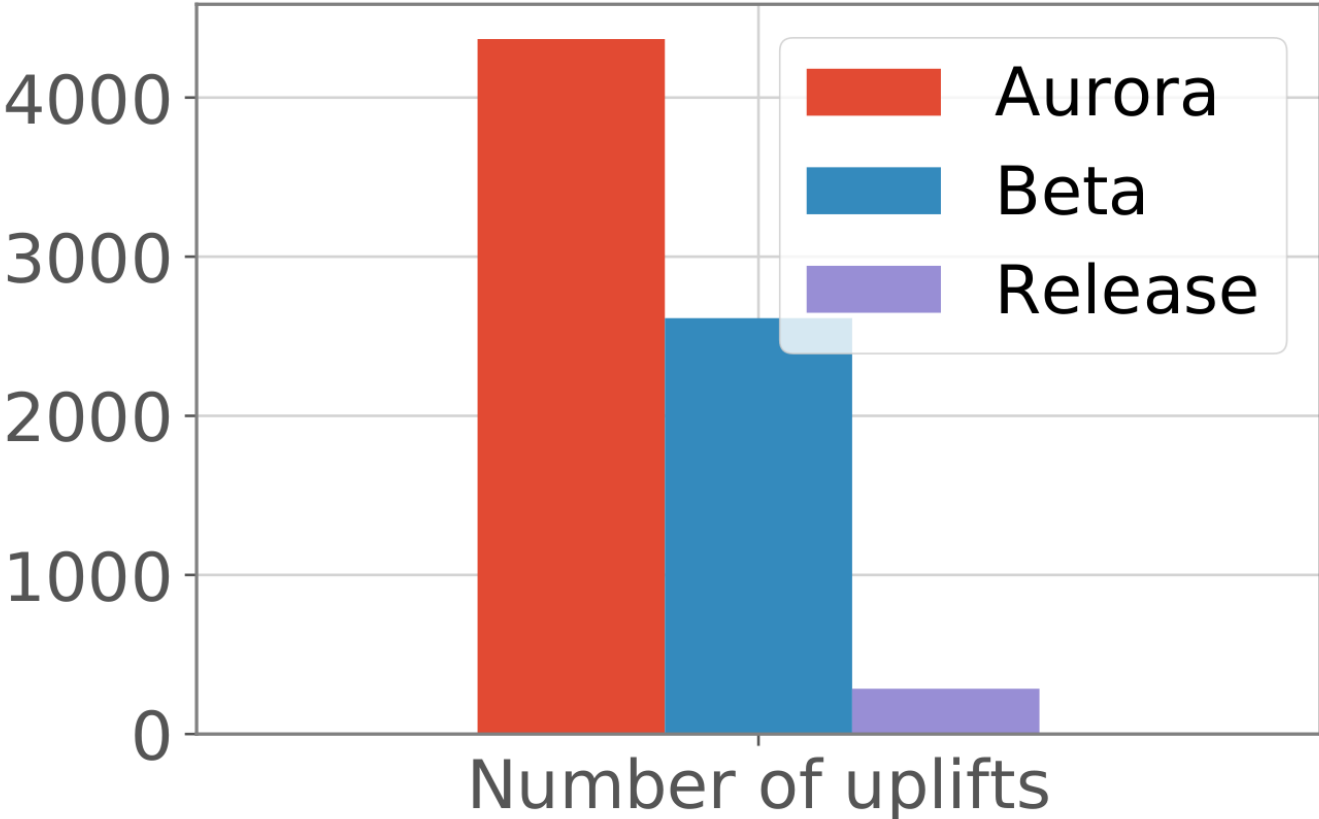


Figure 2: Absolute number of uplifts per channel.

# Empirical study of the uplift process at Mozilla

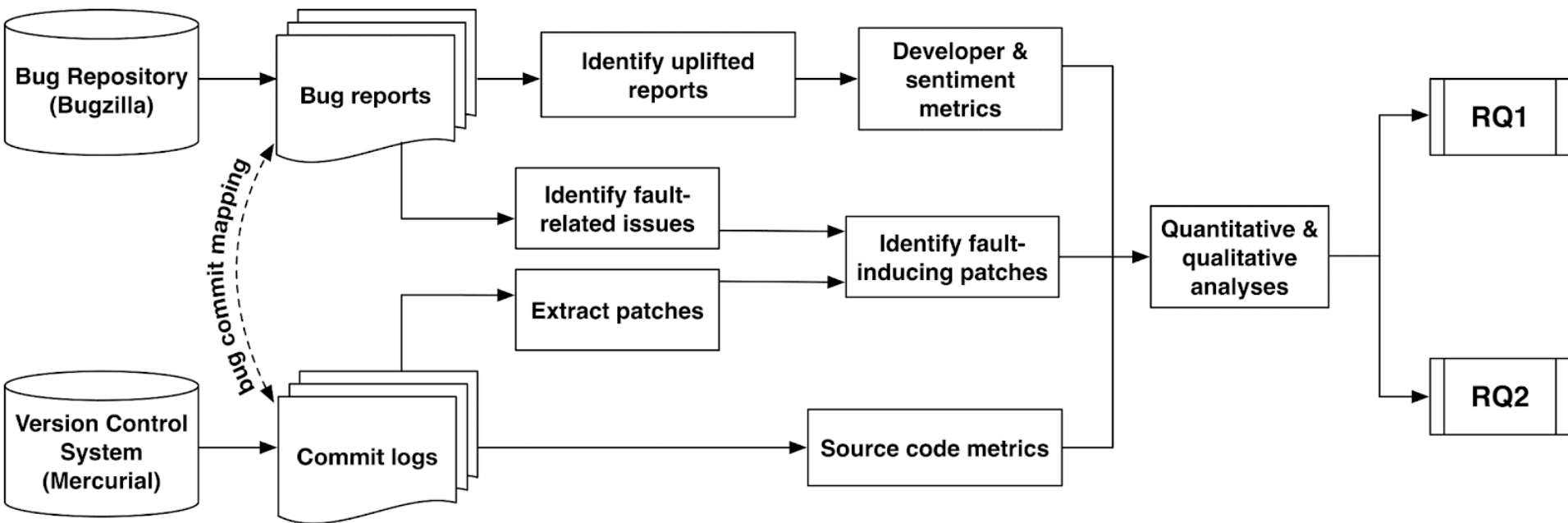
## Methodology

- **Keyword-based heuristic** to identify bugs vs features (categorization that is not available in Bugzilla)
- **SZZ algorithm** to identify fault-inducing patches
- Collected several metrics (developer/reviewer experience and participation; uplift process; sentiment; code complexity) and measured the difference between them in fault-inducing vs non-fault-inducing uplifts
- Manually analyzed a representative set of uplifts and faults to categorize them
- Interviewed release managers for their ideas on the uplift process and their take on our results



# Empirical study of the uplift process at Mozilla

## Methodology



# Empirical study of the uplift process at Mozilla

## Results

- What are the characteristics of patches that are uplifted?
- We observed that most patches are uplifted to resolve wrong functionalities or crashes.
- Rejected uplift requests required longer decision time than accepted requests. We attribute this difference to the high complexity of these rejected patches (since complex patches require longer time for risk assessment).
- Release managers tend to trust patches that concern certain specific components, and—or that are submitted by certain specific developers.

# Empirical study of the uplift process at Mozilla

## Results

- How effective are uplift operations?
- 4% of the subject uplifts did not effectively address the problems but were later reopened, duplicate or cloned into another issue, or required additional uplifts to fix the issue.
- Two major root causes were observed from the ineffective uplifts: the uplifts only partially fixed the issues or caused regressions.
- Higher proportion of ineffective uplifts were detected from the Release channel than from Aurora and Beta.

# Empirical study of the uplift process at Mozilla

## Ineffective uplift types

Category	Description
Not fixed	The issue was completely not fixed, <i>i.e.</i> , the uplifted patch did not have any effect.
Partially fixed	The issue was only partially fixed, <i>i.e.</i> , the uplifted patch had an effect but did not completely resolve the problem.
Need more QA	The uplifted patch had not gone through enough manual verification.
Need more tests	There were no tests added with the uplifted patch, but they were required.
Diagnostics	An uplift was made to gather more data on a problem, then another uplift was made to actually fix it.
Regressions	The uplifted patch caused other defects.
Test failure	The uplifted patch did not pass a certain test.
Build failure	The uplifted patch caused a build error.
Other	Other reasons, <i>e.g.</i> , an issue was fixed by an uplift, but then appeared again because of another patch; or the patch depended on other patches to be uplifted first.

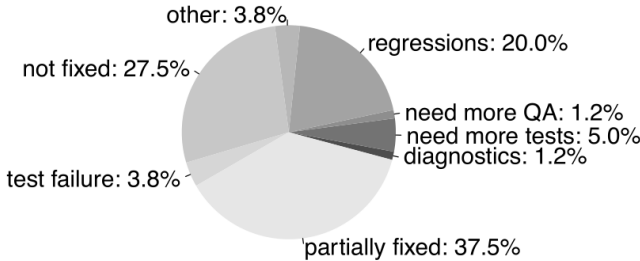
# Empirical study of the uplift process at Mozilla

## Ineffective uplift numbers

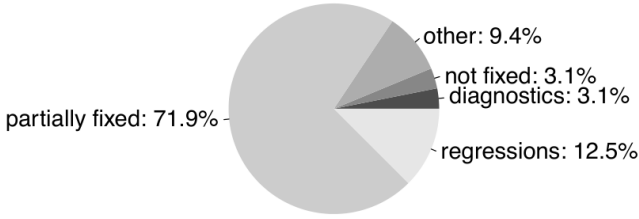
	Aurora	Beta	Release	Unique count
Reopened	70	49	10	77
Cloned	28	16	3	32
Duplicate created after an uplift	15	10	2	16
Duplicate resolved after an uplift	5	3	2	7
Resolved by mul- tiple uplifts	50	42	3	78

# Empirical study of the uplift process at Mozilla

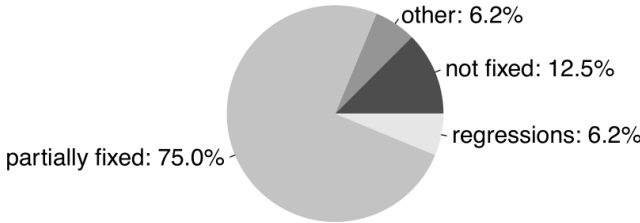
## Root causes of ineffective uplifts



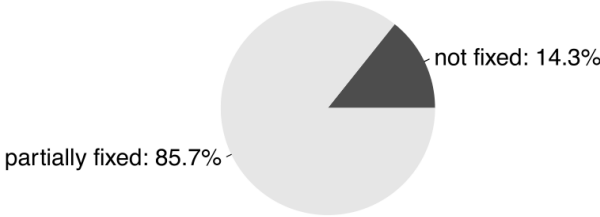
(a) Reopened



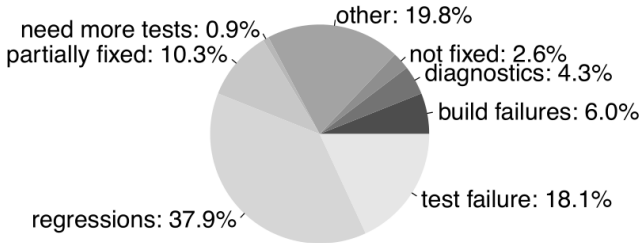
(b) Cloned



(c) Duplicate created after an uplift



(d) Duplicate resolved after an uplift



(e) Resolved by multiple uplifts

# Empirical study of the uplift process at Mozilla

## Results

- Code complexity measures were found to be meaningfully different between fault-inducing and non-fault-inducing uplifts, in particular fault-inducing changes were more likely to have higher complexity (the size of the changes in terms of changed lines, “code churn” in the table, was the main factor).

# Empirical study of the uplift process at Mozilla

## Results

Channel	Metric	Faulty	Clean	<i>p</i> -value	Effect size
<i>Aurora</i>	Code churn	155.0	34.0	<b>5.59e-65</b>	large
	Prior changes	362.5	164.0	<b>3.80e-10</b>	small
	LOC	903.6	457.4	<b>2.23e-06</b>	small
	Cyclomatic	2.5	2.0	<b>1.08e-06</b>	small
	# of functions	34.3	17.0	<b>2.25e-06</b>	small
	Max. nesting	2.7	2.0	<b>5.14e-04</b>	negligible
	Comment ratio	0.2	0.1	<b>4.00e-15</b>	small
	Module number	2.0	1.0	<b>2.99e-24</b>	small
	Closeness	1.5	1.2	<b>2.78e-13</b>	small
	Betweenness	45,221.9	880.7	<b>2.65e-14</b>	small
	PageRank	1.7	1.4	<b>1.95e-15</b>	small
	# of comments	26.0	20.0	<b>1.76e-09</b>	small
	Developer exp.	28.5	10.0	<b>1.19e-18</b>	small
	Reviewer exp.	9.0	2.0	<b>6.63e-09</b>	small
	Comment words	10.0	2.0	<b>9.08e-07</b>	small
	Developer senti.	-3	-3	<b>8.92e-04</b>	negligible
	Owner sentiment	-2	-1	<b>1.66e-04</b>	negligible



# Empirical study of the uplift process at Mozilla

## Results

Channel	Metric	Faulty	Clean	<i>p</i> -value	Effect size
<i>Beta</i>	Code churn	141.0	32.0	<b>6.44e-33</b>	large
	Prior changes	268.0	156.5	<b>1.02e-03</b>	small
	LOC	895.5	476.3	<b>1.66e-03</b>	small
	Cyclomatic	2.5	2.0	<b>3.69e-03</b>	small
	# of functions	37.0	18.0	<b>3.13e-03</b>	small
	Max. nesting	2.7	2.2	<b>0.01</b>	negligible
	Comment ratio	0.2	0.1	<b>4.61e-05</b>	small
	Module number	2.0	1.0	<b>7.45e-12</b>	small
	Closeness	1.6	1.2	<b>2.87e-07</b>	small
	Betweenness	35,661.7	1,327.8	<b>6.00e-08</b>	small
	PageRank	1.7	1.4	<b>1.08e-06</b>	small
	# of comments	28.0	22.0	<b>1.18e-04</b>	small
	Comment words	8.0	3.0	<b>0.04</b>	negligible
	Developer exp.	29.0	10.0	<b>1.33e-08</b>	small
	Reviewer exp.	10.0	2.0	<b>3.35e-05</b>	small
	Owner sentiment	-2	-1	<b>4.14e-03</b>	small

# Empirical study of the uplift process at Mozilla

## Results

Channel	Metric	Faulty	Clean	<i>p</i> -value	Effect size
<i>Release</i>	Code churn	108.0	27.0	<b>2.07e-03</b>	large

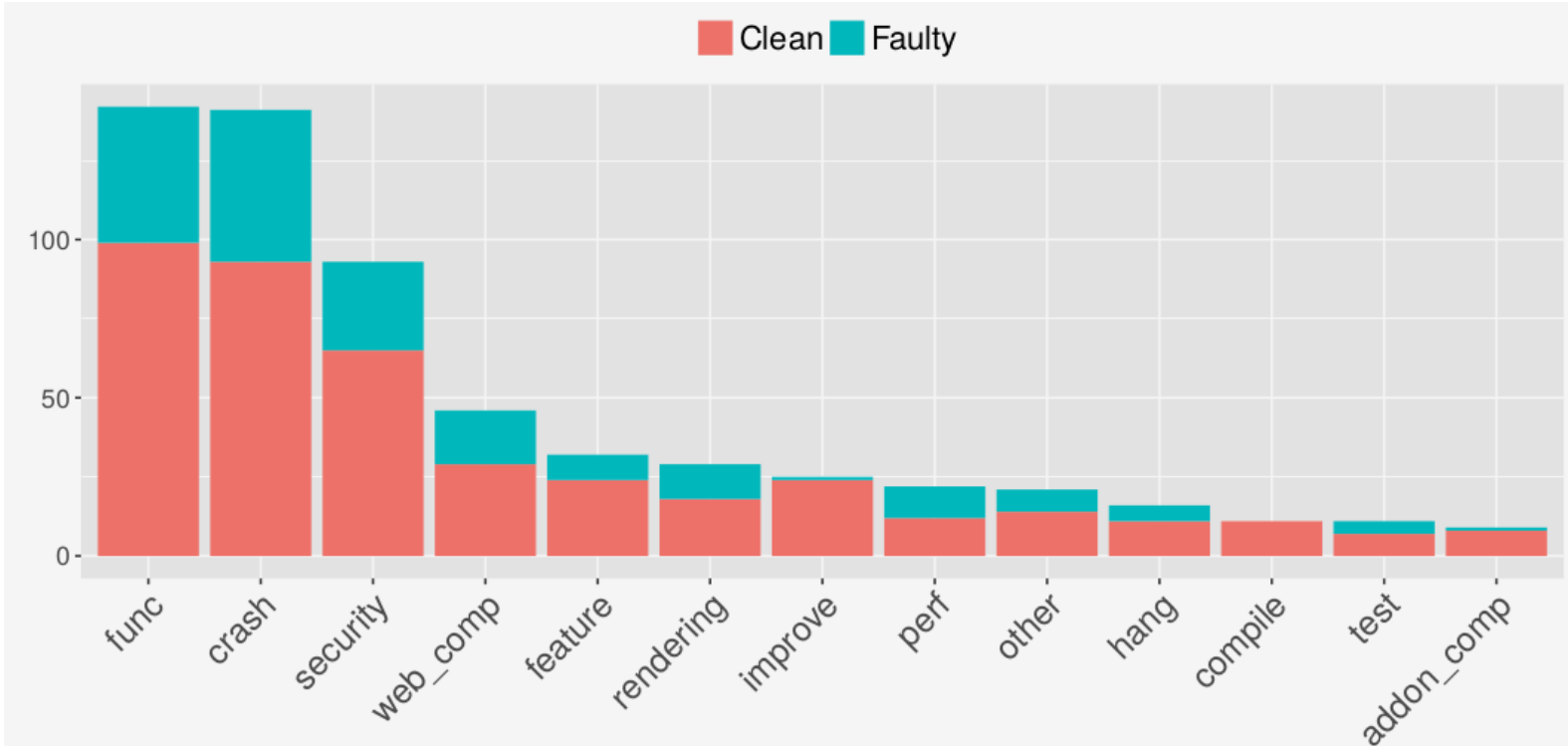
# Empirical study of the uplift process at Mozilla

## Reasons for uplift

Reason	Description
Security	Security vulnerability.
Crash	Program unexpectedly stops running.
Hang	Program keeps running but without response.
Performance degradation (perf)	Functionalities are correct but response is slow or delayed.
Incorrect rendering (rendering)	Components or video cannot be correctly rendered.
Wrong functionality (func)	Incorrect functionalities besides rendering issues.
Web incompatibility (web comp)	Program does not work correctly for a major website or many websites due to incompatible APIs or libraries, or a functionality, which was removed on purpose, but is still used in the wild.
Add-on or plug-in incompatibility (addon comp)	Program does not work correctly for a major add-on/plug-in or many add-ons/plug-ins due to incompatible APIs or libraries, or a functionality, which was removed on purpose, but is still used in the wild.
Compile	Compiling errors.
Feature	Introduce or remove features.
Improvement (improve)	Minor functional or aesthetical improvement.
Test-only problem (test)	Errors that only break tests.
Other	Other uplift reasons, <i>e.g.</i> , data corruption and license incompatibility.

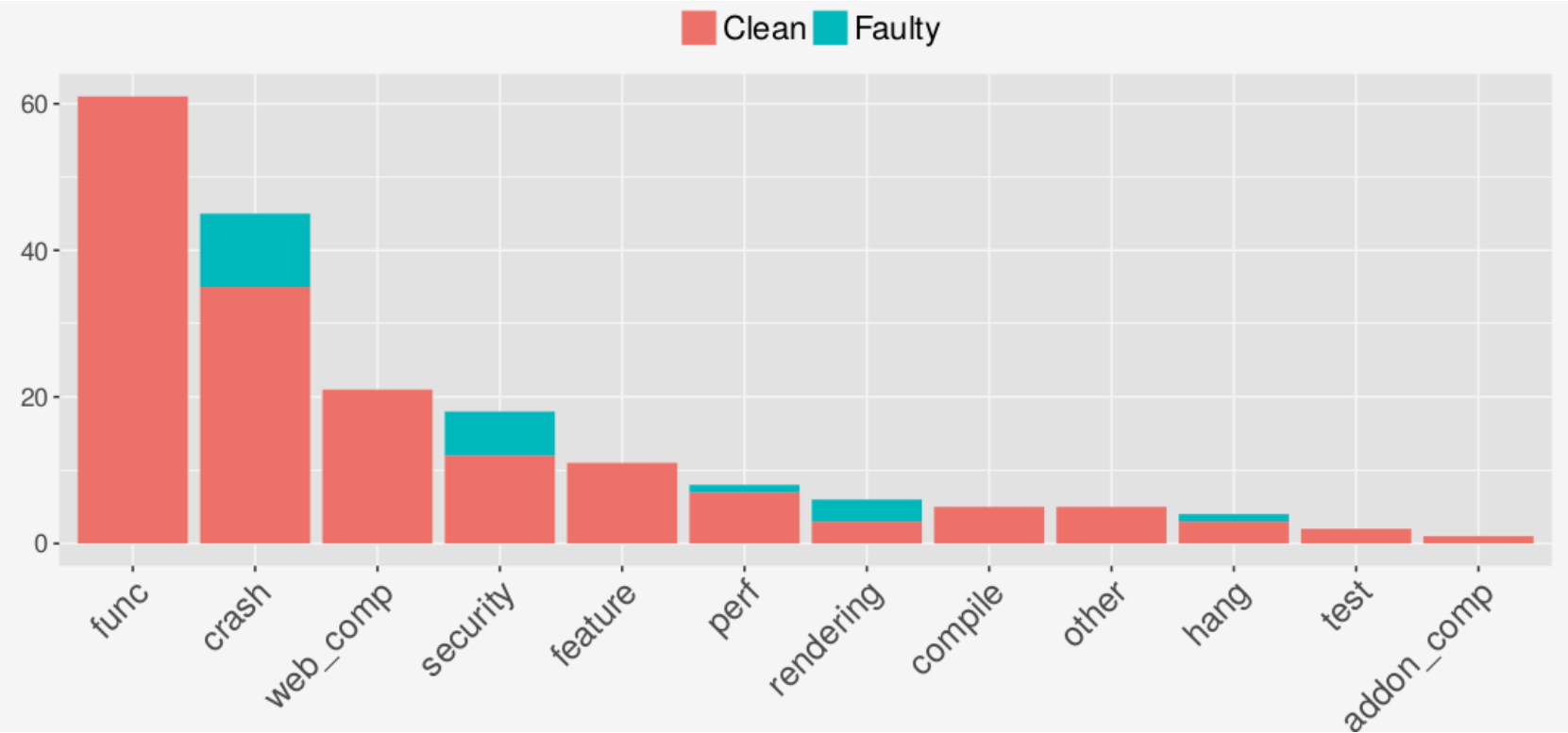
# Empirical study of the uplift process at Mozilla

## Reasons for uplift in Beta



# Empirical study of the uplift process at Mozilla

## Reasons for uplift in Release



# Empirical study of the uplift process at Mozilla

## Reasons for uplifts

- Performance uplifts are quite risky and so they should be considered more carefully for uplift.
- Uplifts to fix incorrect functionality or web compatibility problems are more likely non-fault-inducing in release than in beta. A possible explanation is that release uplifts are tested more thoroughly.

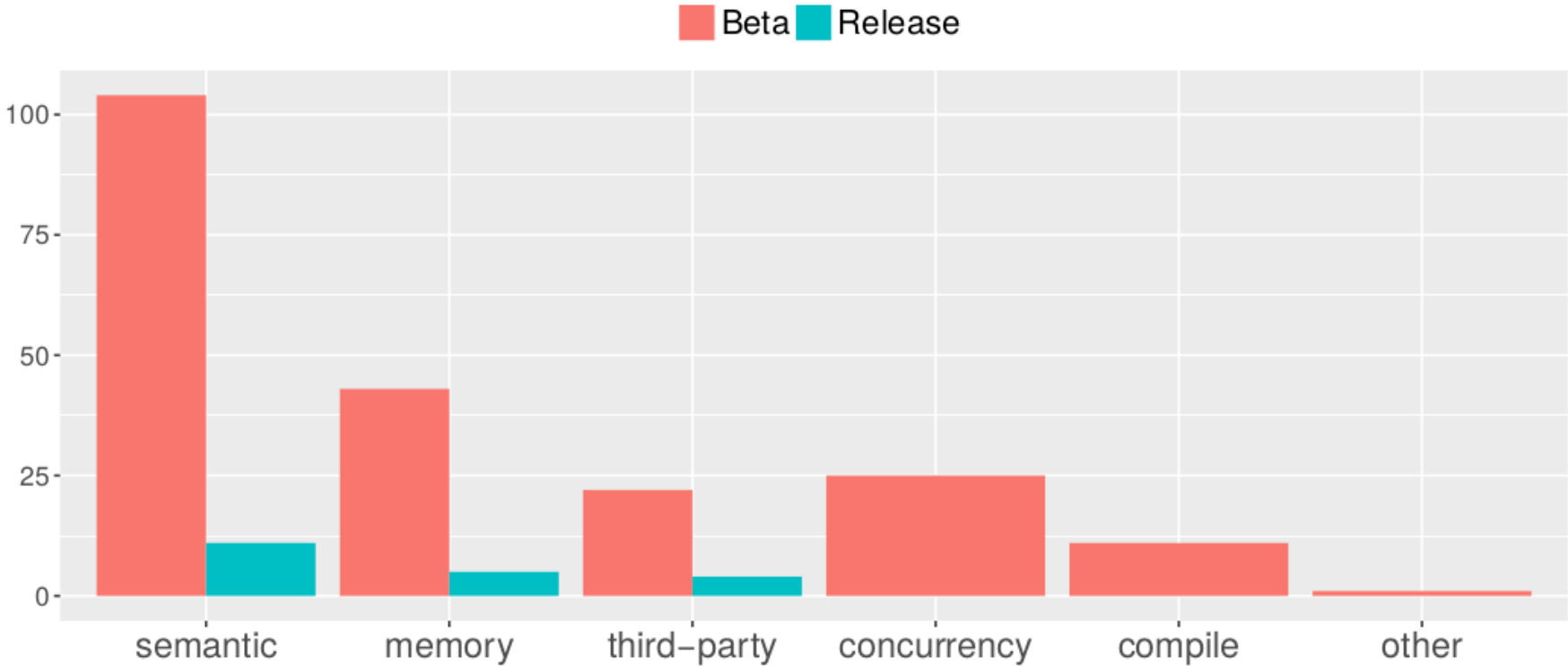
# Empirical study of the uplift process at Mozilla

## Classification of faults introduced by uplifts

Reason	Description
Memory	Memory errors, including memory leak, overflow, null pointer dereference, dangling pointer, double free, uninitialized memory read, and incorrect memory allocation.
Semantic	Semantic errors, including incorrect control flow, missing functionality, missing cases of a functionality, missing feature, incorrect exception handling, and incorrect processing of equations and expressions.
Third-party	Errors due to incompatibility of drivers, plug-ins or add-ons.
Concurrency	Synchronization problems between multiple threads or processes, <i>e.g.</i> , incorrect mutex usage.
Compile	Compile-time errors.
Other	Other errors.

# Empirical study of the uplift process at Mozilla

## Faults introduced by uplifts





# Empirical study of the uplift process at Mozilla

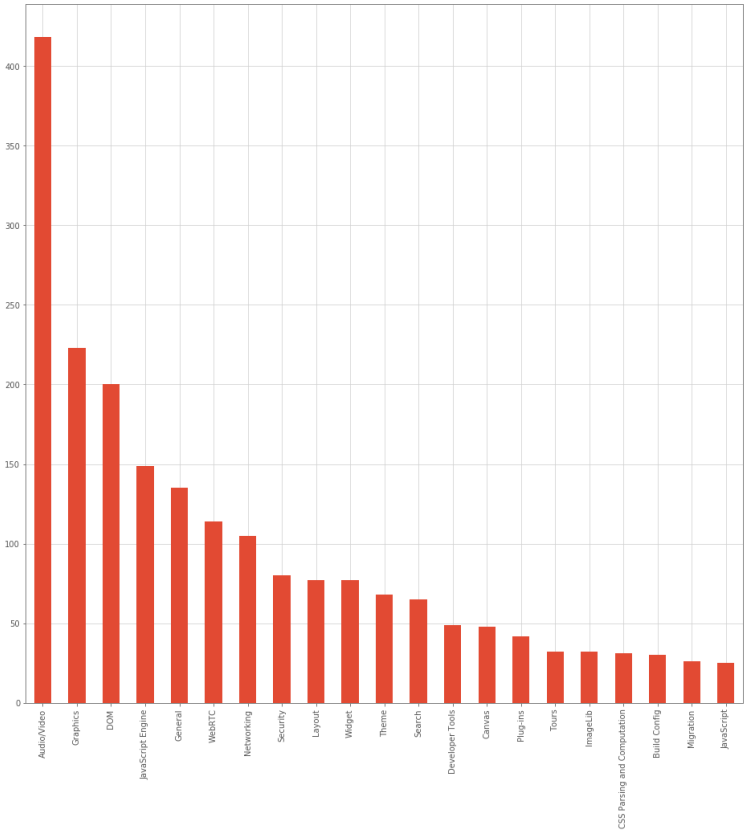
## Classification of faults introduced by uplifts

- The results show that a sizeable number of faults introduced by uplifts could potentially be prevented by static analysis / safer languages.
- Semantic and third-party faults can only be prevented by more thorough testing.

# Empirical study of the uplift process at Mozilla

Number of uplifts per component (Beta)

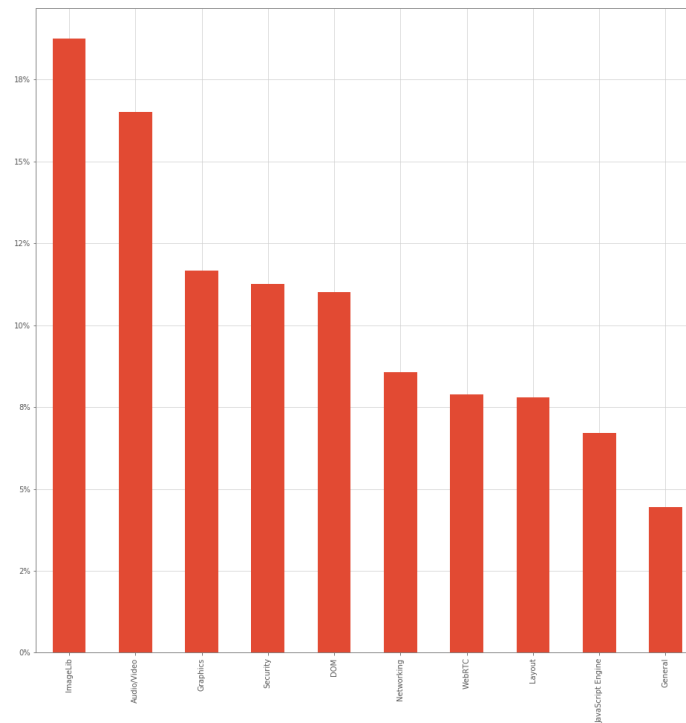
**Uplifts are more common in specific components.**



# Empirical study of the uplift process at Mozilla

## Faults-proneness per component (Beta)

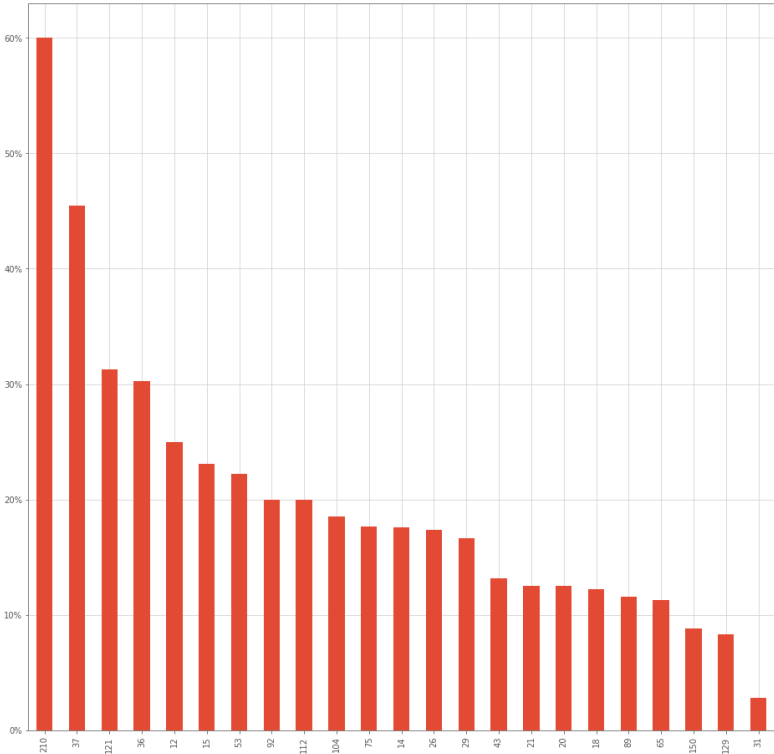
**Uplifts in some components are more likely to cause regressions.**



# Empirical study of the uplift process at Mozilla

## Faults-proneness per developer (Beta)

Uplifts requested by different developers have different probabilities of causing regressions.



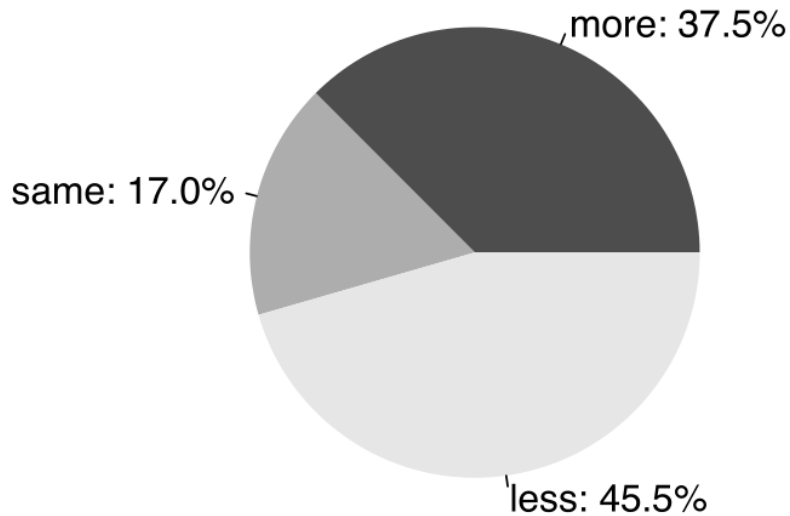
# Empirical study of the uplift process at Mozilla

## Results

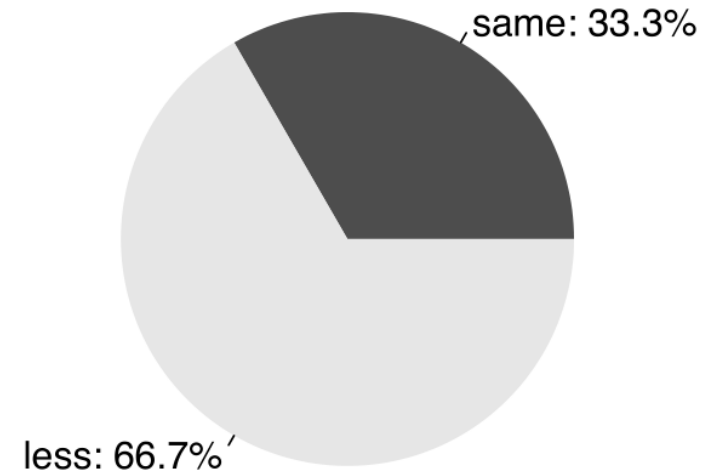
- Are regressions caused by uplift more severe than the bugs that were fixed with the uplift?
- Through a manual analysis, we observed that 37.5% of the Beta fault-inducing uplifts caused a “more severe regression”, i.e., regression that is more severe than the problems they aimed to address.
- No “more severe regression” was found from the examined Release uplifts, perhaps due to a more strict uplift policy and code review process on this channel.

# Empirical study of the uplift process at Mozilla

Whether the regression an uplift caused is more severe than the problem the uplift aims to address.



(a) Beta channel



(b) Release channel

# Empirical study of the uplift process at Mozilla

## Uplift -> Regression transitions (Beta)

Uplift	Regression	Frequency	Probability
<i>compile</i>	<i>crash</i>	2	0.67
compile	compile	1	0.33
crash	crash	24	0.50
crash	func	13	0.27
crash	compile	5	0.10
crash	perf	3	0.06
crash	other	2	0.04
<i>crash</i>	<i>security</i>	1	0.02
func	func	35	0.57
<i>func</i>	<i>crash</i>	14	0.23
func	perf	7	0.11
func	compile	4	0.07
func	other	1	0.02
<i>improve</i>	<i>crash</i>	7	0.37
<i>improve</i>	<i>func</i>	7	0.37
improve	compile	2	0.11
<i>improve</i>	<i>perf</i>	2	0.11
<i>improve</i>	<i>security</i>	1	0.05
<i>perf</i>	<i>func</i>	5	0.50
<i>perf</i>	<i>crash</i>	4	0.40
perf	perf	1	0.10
security	func	8	0.33
security	crash	7	0.29
security	security	5	0.21
security	compile	2	0.08
security	other	1	0.04
security	perf	1	0.04

# Empirical study of the uplift process at Mozilla

Uplift -> Regression transitions (Release)

Uplift	Regression	Frequency	Probability
crash	func	6	0.55
crash	crash	5	0.45
func	func	1	0.50
func	perf	1	0.50
security	func	2	0.50
security	security	2	0.50



# Empirical study of the uplift process at Mozilla

## Results

- Could some of the regressions have been prevented through more extensive testing on the channels?
- We considered regressions to be possibly preventable if they were reproducible not only by the issue reporter and were found either on a widely used feature/website/configuration or via Mozilla's telemetry
- We manually examined a sample of regressions due to Beta and Release uplifts
- 25% of the regressions due to Beta uplifts and 30% of the regressions due to Release uplifts could have been possibly prevented.

# Empirical study of the uplift process at Mozilla

By whom a regression was reproducible

<b>Reproducible</b>	<b>Description</b>
By all	Everybody was able to reproduce.
By some	Somebody was able to reproduce (depending for example on the version of a driver, or a specific version of an operating system, and so on).
By the reporter only	Nobody else except the reporter was able to reproduce.
By no one	Nobody was able to reproduce (and the issue was found, for example, by analyzing crash reports).

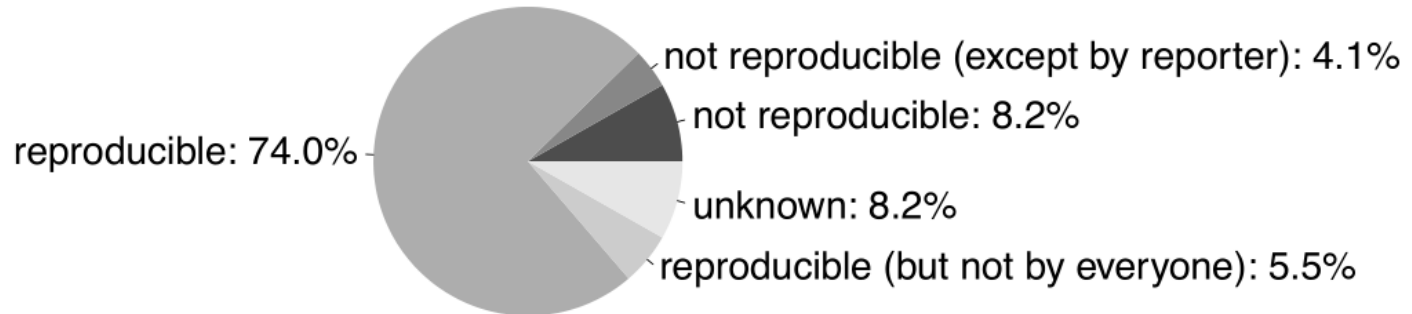
# Empirical study of the uplift process at Mozilla

## How regressions were found

Found	Description
By tooling	The issue was found by fuzzing or static analysis.
By developers	The issue was found by Mozilla developers (by code inspection, by running tests that were not included in Firefox' test suites, or by running special tools such as Valgrind or ASan) or by an external developer ( <i>e.g.</i> , a security researcher).
On a widely used feature/website/-config	The issue was found by a user (an end-user, a volunteer, or a website developer) on a widely used feature, on a widely used website, or in a widespread configuration.
On a rarely used feature/website/-config	The issue was found by a user on a rarely used feature or rarely used website or on an uncommon configuration.
Via telemetry	The issue was found by analyzing crash reports or performance measurements from the field.

# Empirical study of the uplift process at Mozilla

Whether the regressions caused by an uplift were reproducible



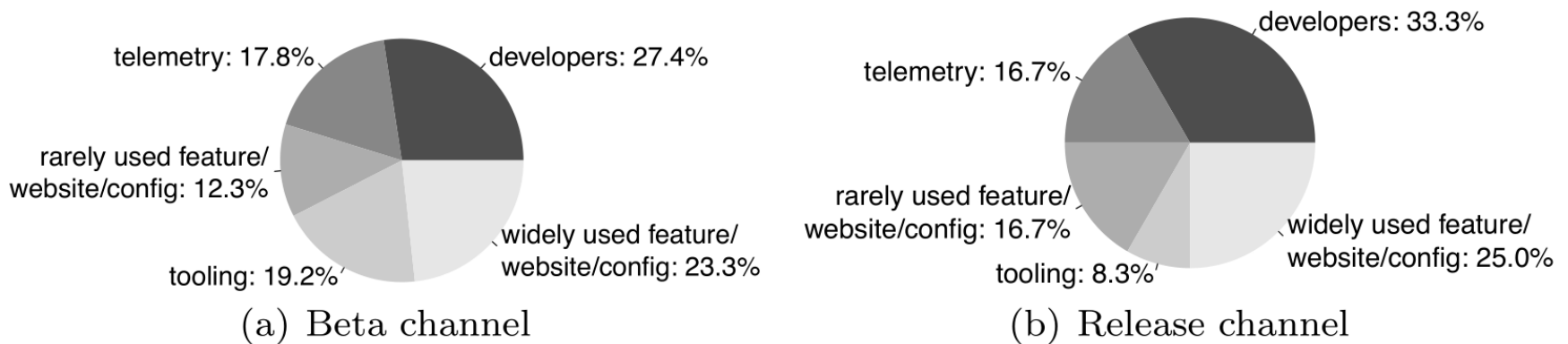
(a) Beta channel



(b) Release channel

# Empirical study of the uplift process at Mozilla

How the regressions caused by uplifts were found



# Understanding Flaky Tests: Relevance, Nature, and Challenges

## Understanding Flaky Tests: Relevance, Nature, and Challenges

- Around 7'000 tests per week fail intermittently
- They sneakily decrease the value of the test suites
- They decrease developers' trust in test suites
- They make it more difficult to notice real regressions

# Understanding Flaky Tests: Relevance, Nature, and Challenges

## Overview

- Empirical investigation on software repository data (pertaining to 391 software systems)
- A novel dataset of 200 flaky tests classified by practitioners who fixed those tests
- The opinions of 120 developers collected in an online questionnaire



## Understanding Flaky Tests: Relevance, Nature, and Challenges

### First research question

- How prominent is test flakiness and how problematic is it as perceived by developers?
- The mining study and the collected developers' opinions indicate that flaky tests are rather frequent and a non-negligible problem, with possibly important consequences on resource allocation and scheduling, as well as on the reliability of the test suite.

**Understanding Flaky Tests:  
Relevance, Nature, and Challenges**  
Second research question

- How can the causes of flaky tests be categorized?
- We confirm the existence of seven flakiness types revealed by Luo et al.. We discover four additional categories, three of which developers consider as the most effort-prone types of flakiness to deal with. Finally, we provide evidence that flaky tests can be also due to problems in production code.

# Understanding Flaky Tests: Relevance, Nature, and Challenges

## Second research question

	Effort	Origin
Concurrency	4.0	.66 T
Async Wait	3.0	1.00 T
Assertion Failure	1.0	1.00 T
Test Order	2.0	1.00 T
Test Case Timeout	4.0	1.00 T
Resource Leak	3.0	.85 T
Platform Dependency	4.0	.90 T
Float Precision	4.0	1.00 T
Test Suite Timeout	3.5	1.00 T
Time	1.0	1.00 T
Randomness	1.0	1.00 T

**Understanding Flaky Tests:  
Relevance, Nature, and Challenges**  
Third research question

- What are the challenges that developers face when dealing with flaky tests?
- Reproducing the failing context, understanding nature and elements involved in the flakiness, and knowing whether the flakiness is originated by test or production code are the most serious challenges for developers. Moreover, designing test code in a proper manner is an additional major challenge not mentioned in the reviewed academic and gray literature.

# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

- Between 2015-07-02 and 2017-08-25, out of 15 Firefox releases, 8 (46, 48, 49, 50, 52, 53, 55, 56) have been «blocked» because of a bug caused by a third-party software.

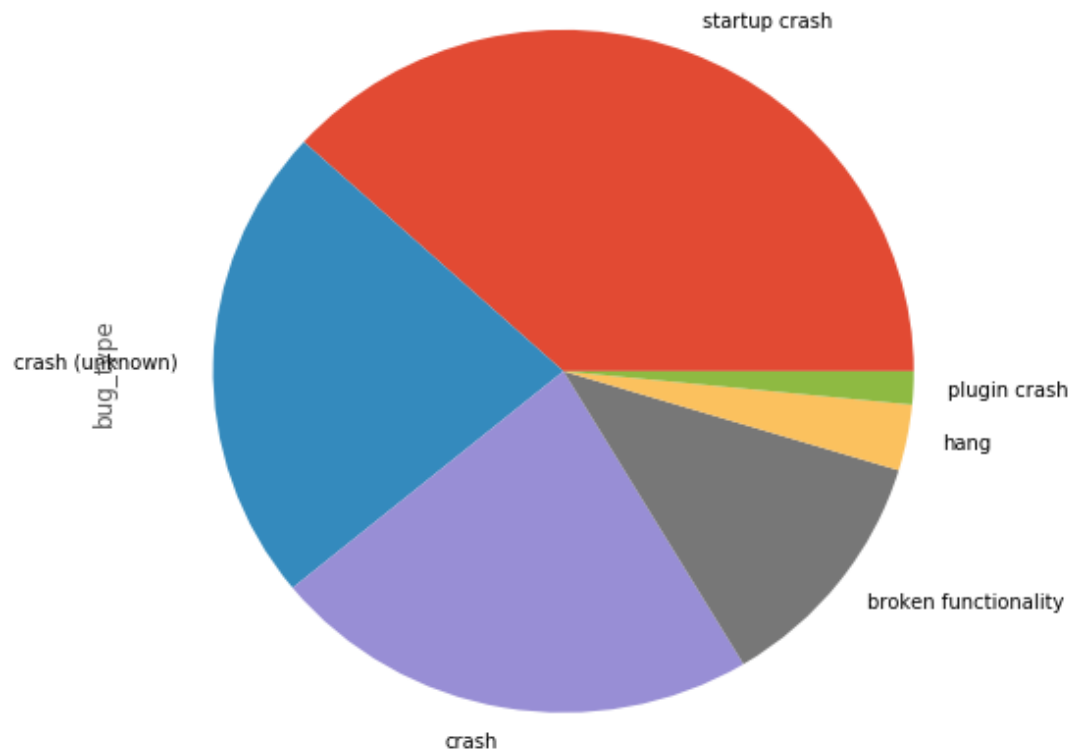
# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

## Current process

- Contact the third-party software developers to notify them of the problem
- Try to reproduce the problem
- Try to block the software with a blocklist addition (which requires a new Firefox build and shipping an update to users)

# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

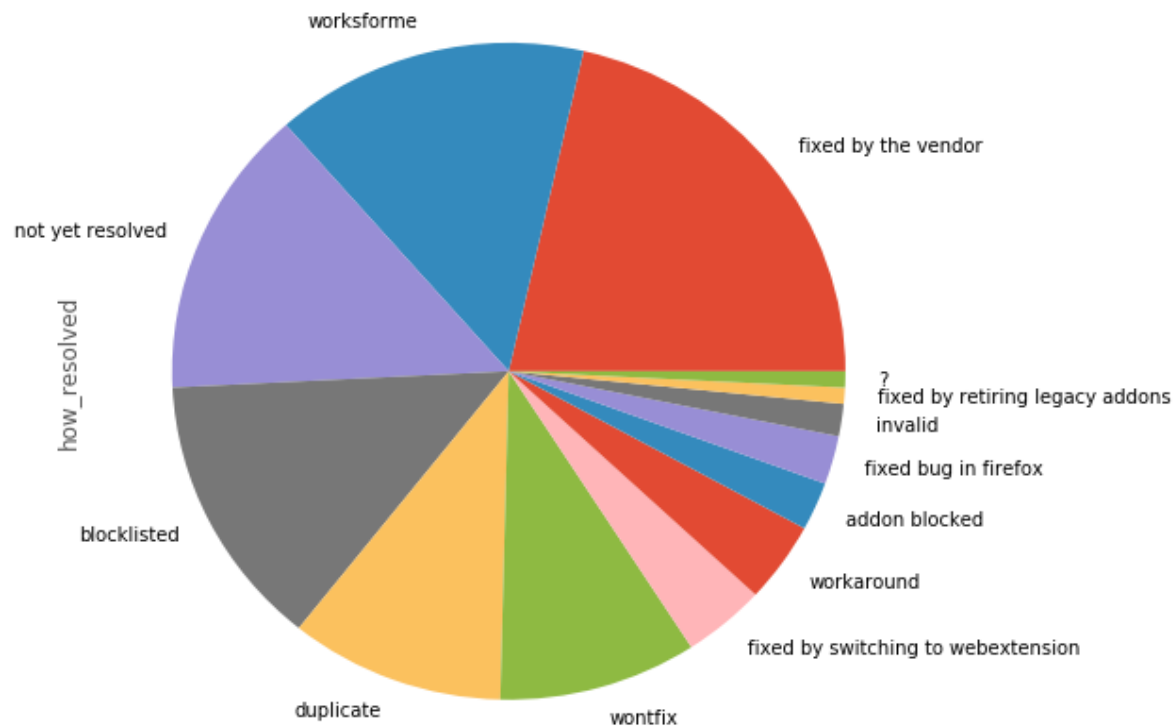
## Type of bug





# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

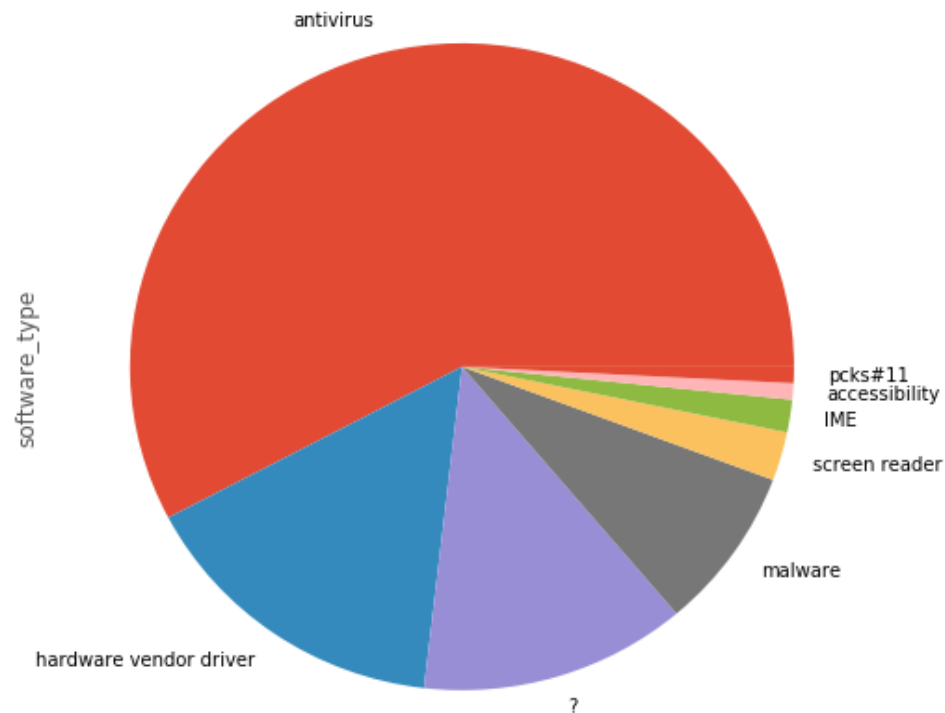
## How the bug was resolved





# An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem

## Type of the third-party software



# **An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem**

Survey with third-party software developers

- The majority of them don't perform any kind of QA
- They use «hacky» techniques, instead of public and vetted APIs, to inject their code in other processes

# **An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem**

Possible solution: blacklist and whitelist

- Force developers to perform QA in order to be allowed in the whitelist
- Remove software from the whitelist when it causes problems
- Offer preferential access to the whitelist for software which uses public and vetted APIs
- Improve the APIs to allow additional use cases

# Detecting web compatibility issues using CNNs

# Detecting web compatibility issues using CNNs

## Objectives

- Automatically detect web compatibility issues
- Automatically detect regressions after large refactorings (e.g. recently, after the introduction of a new style engine)
- Offer a tool for web developers to automatically test their websites for compatibility

# Detecting web compatibility issues using CNNs

## Web Compatibility

- Issues that present themselves only in a certain class of browser/systems (usually due to usage of unstable or non-standard APIs, implemented by a single browser, or corner cases in the specifications, or limited testing, or marketing)



# Detecting web compatibility issues using CNNs

## Web Compatibility



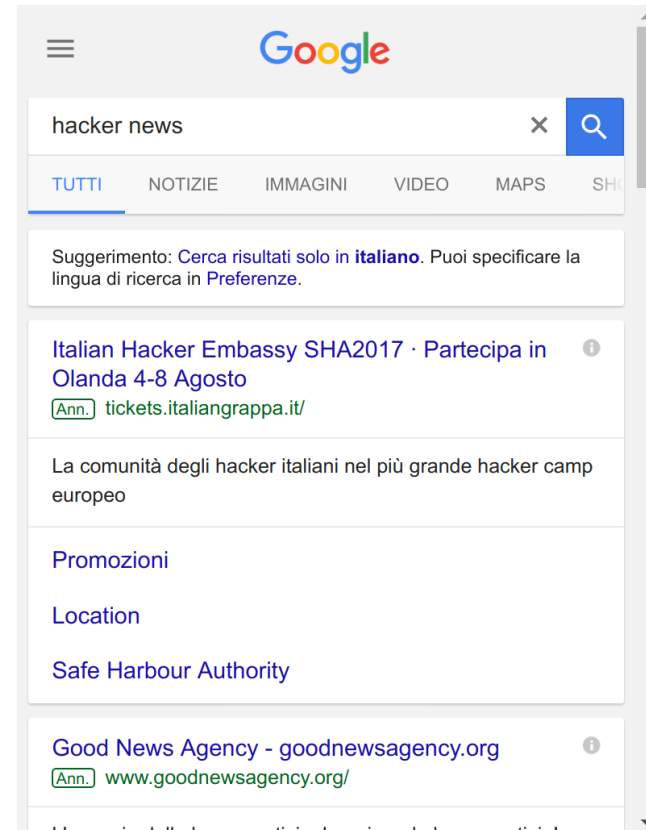
Google hacker news

Tutti Immagini Altro

**Hacker News**  
<https://news.ycombinator.com/>  
Show **HN**: Haskell library for pricing and information on crypto-currencies (github. com). 58 points by aviaviavi 5 hours ...  
News Ycombinator - New - FAQ - Jobs

**The Hacker News — Cyber Security, Hacking News**  
<thehackernews.com/?m=1>  
The **Hacker News** — leading source of Information Security, latest **Hacking News**, Cyber Security, Network Security with ...  
(New) Become A Professional ... - Tech - Learn Ethical Hacking Online - Data Breach

**Hacker News - Reddit**  
<https://www.reddit.com/r/hackernews/>  
**hackernews**. subscribeunsubscribe16,196 readers. 87 users here now. A mirror of **Hacker News**. a community for 9 ...



Google

hacker news

TUTTI NOTIZIE IMMAGINI VIDEO MAPS SH

Suggerimento: Cerca risultati solo in italiano. Puoi specificare la lingua di ricerca in Preferenze.

Italian Hacker Embassy SHA2017 · Partecipa in Olanda 4-8 Agosto  
[Ann.](#) <tickets.italiangrappa.it/>

La comunità degli hacker italiani nel più grande hacker camp europeo

Promozioni

Location

Safe Harbour Authority

Good News Agency - goodnewsagency.org  
[Ann.](#) [www.goodnewsagency.org/](http://www.goodnewsagency.org/)

Il fascino della buona notizia. Leggere la buona notizia

# Detecting web compatibility issues using CNNs

## Web Compatibility

- The compatibility problem is often in the structure of the page, while the content can change (e.g. a news site, ads, a carousel, etc.)

# Detecting web compatibility issues using CNNs

## Overview

- Collection of a dataset of screenshots using Selenium
- Training of a Convolutional Neural Network on couples of images (one from Firefox, one from Chrome) to detect images which are not compatible
- Use of a Siamese architecture, particularly suitable to this kind of problems

# Detecting web compatibility issues using CNNs

## Overview

- Loading of pages from webcompat.com (a tracker of web compatibility issues)
- Random selection of elements of the page to interact with (depth first search), in an attempt to reproduce problems
- Repetition of  $\wedge$  multiple times per day, on different days (to «teach the network» that some web sites are compatible even if their content changes)

# What Makes a Code Change Easier to Review

ESEC/FSE 2018

# What Makes a Code Change Easier to Review

- Interested in finding what makes code easy/difficult to review
- Developed and deployed an add-on on Mozilla review tools (at the moment, four different systems...) to collect feedback about patches from reviewers

# What Makes a Code Change Easier to Review

The screenshot shows a Bugzilla interface for reviewing a code patch. At the top, there's a navigation bar with the Bugzilla logo, a search bar, and links for 'Browse', 'Advanced Search', 'New Bug', and 'My Dashboard'. A notification icon with a red '2' is also present. The main content area displays a code diff for 'js/src/jit-test/tests/coverage/simple.js'. The diff shows changes between lines 101-103 and 104-106 on the left, and 101-103 and 104-106 on the right. A green highlight covers the code between lines 104 and 117. Below the code, there's a 'Powered by Splinter' logo. A feedback section follows, starting with five stars and the question 'How would you rate the **reviewability** of this patch?'. Below this are two text input fields: 'How long did you take to review this patch?' (with a 'Number of minutes' dropdown) and 'What aspects of this patch, if any, contribute to its reviewability?'. A second field asks 'What aspects of this patch, if any, should be improved to enhance its reviewability?'. A 'Submit feedback' button is located at the bottom right of the form.

# What Makes a Code Change Easier to Review

## Findings

- Interviewed developers primarily focus on three aspects for reviewability:
  - code churn;
  - change description;
  - commit history
- Surprisingly, test inclusion does not play a significant role in reviewability
- The connection between reviewability and acceptance is weak



# What Makes a Code Change Easier to Review

## Findings

- Interviewed developers primarily focus on three aspects for reviewability:
  - code churn;
  - change description;
  - commit history
- Surprisingly, test inclusion does not play a significant role in reviewability
- The connection between reviewability and acceptance is weak

# Publications

- M. Castelluccio, G. Poggi, C. Sansone, L. Verdoliva – Land Use Classification in Remote Sensing Images by Convolutional Neural Networks – <https://arxiv.org/abs/1508.00092>
- M. Castelluccio, G. Poggi, C. Sansone, L. Verdoliva – Training Convolutional Neural Networks for Semantic Classification of Remote Sensing Imagery – JURSE2017 (submitted)

# Publications

- M. Castelluccio, C. Sansone, L. Verdoliva, and G. Poggi – Automatically analyzing groups of crashes for finding correlations – Proceedings of the 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2017)
- M. Castelluccio, L. An, and F. Khomh – Is It Safe to Uplift This Patch? An Empirical Study on Mozilla Firefox – In proceedings of the 33rd International Conference on Software Maintenance and Evolution (ICSME 2017). Received IEEE TCSE Distinguished Paper Award. Invited for publication on “Empirical Software Engineering” journal.

# Publications

- A. Ram, A.A. Sawant, M. Castelluccio, and A. Bacchelli – What Makes A Code Change Easier to Review? An Empirical Investigation On Code Change Reviewability – Proceedings of the 2018 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)
- L. An, F. Khomh, S. McIntosh, and M. Castelluccio – Why Did This Reviewed Code Crash? An Empirical Study of Mozilla Firefox – The 25th Asia-Pacific Software Engineering Conference (APSEC 2018)

# Publications

- M. Castelluccio, L. An, and F. Khomh – An Empirical Study of Patch Uplift in Rapid Release Development Pipelines – Empirical Software Engineering journal (EMSE)
- M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli – Understanding Flaky Tests: Relevance, Nature, and Challenges – ICSE2019 (submitted)

# Publications

- L. An, M. Castelluccio, and F. Khomh – An Empirical Study of DLL Injection Bugs in the Firefox Ecosystem – EMSE (submitted)

# Questions

# Previous Studies on Urgent Patches

- S. Hassan, W. Shang, and A. E. Hassan, “*An empirical study of emergency updates for top android mobile apps*”, Empirical Software Engineering
- D. Lin, C.-P. Bezemer, and A.E. Hassan, “*Studying the urgent updates on the steam platform*”  
Engineering
- M. T. Rahman and P. ...  
*stabilization on linux and chrome*”, IEEE Software

*None of these studies has empirically investigated how urgent patches affect software quality in terms of fault-proneness*



# Case Study Design - Subject System

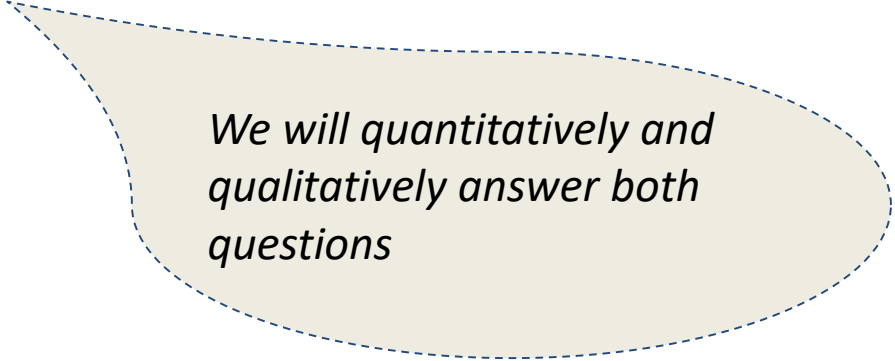
Mozilla Firefox because:

- Mozilla is the most studied system for the rapid release problem
- Mozilla's uplift data is publicly available
- One of our authors works at Mozilla and can easily interview Mozilla release managers

# Case Study Design - Research Questions

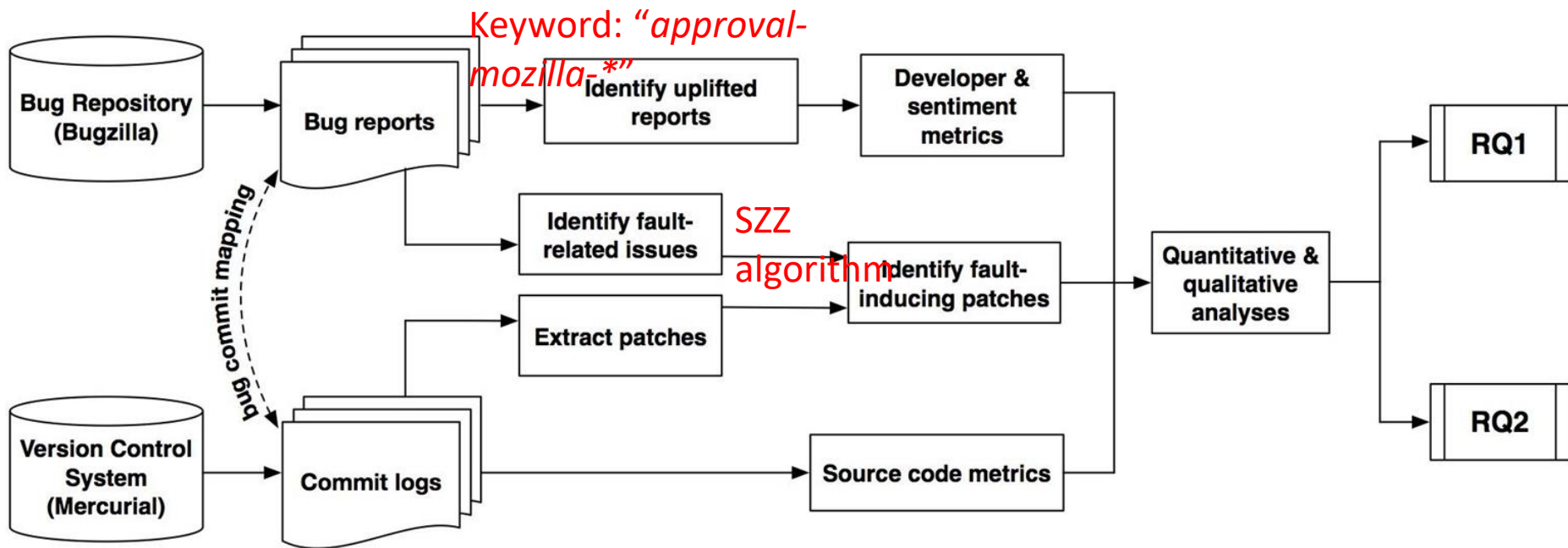
RQ1: What are the characteristics of patches that are uplifted?

RQ2: What are the characteristics of uplifted patches that introduced faults in Mozilla Firefox?



*We will quantitatively and qualitatively answer both questions*

# Case Study Design - Analysis Overview



# Case Study Design

## (Quantitative Analysis - Metrics)

### **Experience & participation**

*developer experience, reviewer experience, comment number, comment words, review duration*

### **Uplift process**

*landing delta, response delta, release delta*

### **Sentiment**

*developer sentiment, module owner sentiment*

### **Social network analysis**

*PageRank, betweenness, closeness*

### **Code complexity**

*patch size, test patch size, prior changed times, LOC, McCabe, function number, max. Nesting, comment ratio, module number*

# Case Study Design

## (Quantitative Analysis - Statistical Tests)

We use the *Mann-Whitney U test* ( $\alpha = 0.05$ ) to investigate whether there is a statistically significant difference between a patch that was uplifted and a patch that was not uplifted (RQ1), and whether there is a statistically significant difference between a faulty uplifted patch and a clean uplifted patch (RQ2).

For the results with statistically significant difference, we will use *cliff's delta* to measure the magnitude of the difference (*i.e.*, effect size).

# Case Study Design

## (Qualitative Analysis - RQ1)

Based on a random sampling, we manually examine and classify the *reasons why developers uplift patches* in the Beta and Release channels.

We interview Mozilla release managers on the following questions:

- 1) “Which factors do you take into account when deciding about an uplift?”
- 2) “Are there differences in how you handle uplifts in different channels, and what are the differences?”
- 3) “How do you decide which developers you can trust?”

# Case Study Design

## (Qualitative Analysis - RQ2)

Based on a random sampling, we manually examine and classify the *root causes of uplifted patches that lead to faults* in the Beta and Release channels.

We interview Mozilla release managers on the following question:

“What are the characteristics of the fault-introducing patches that you are not currently taking enough into account but could be considered in the future?”

# Results - RQ1 (statistical tests)

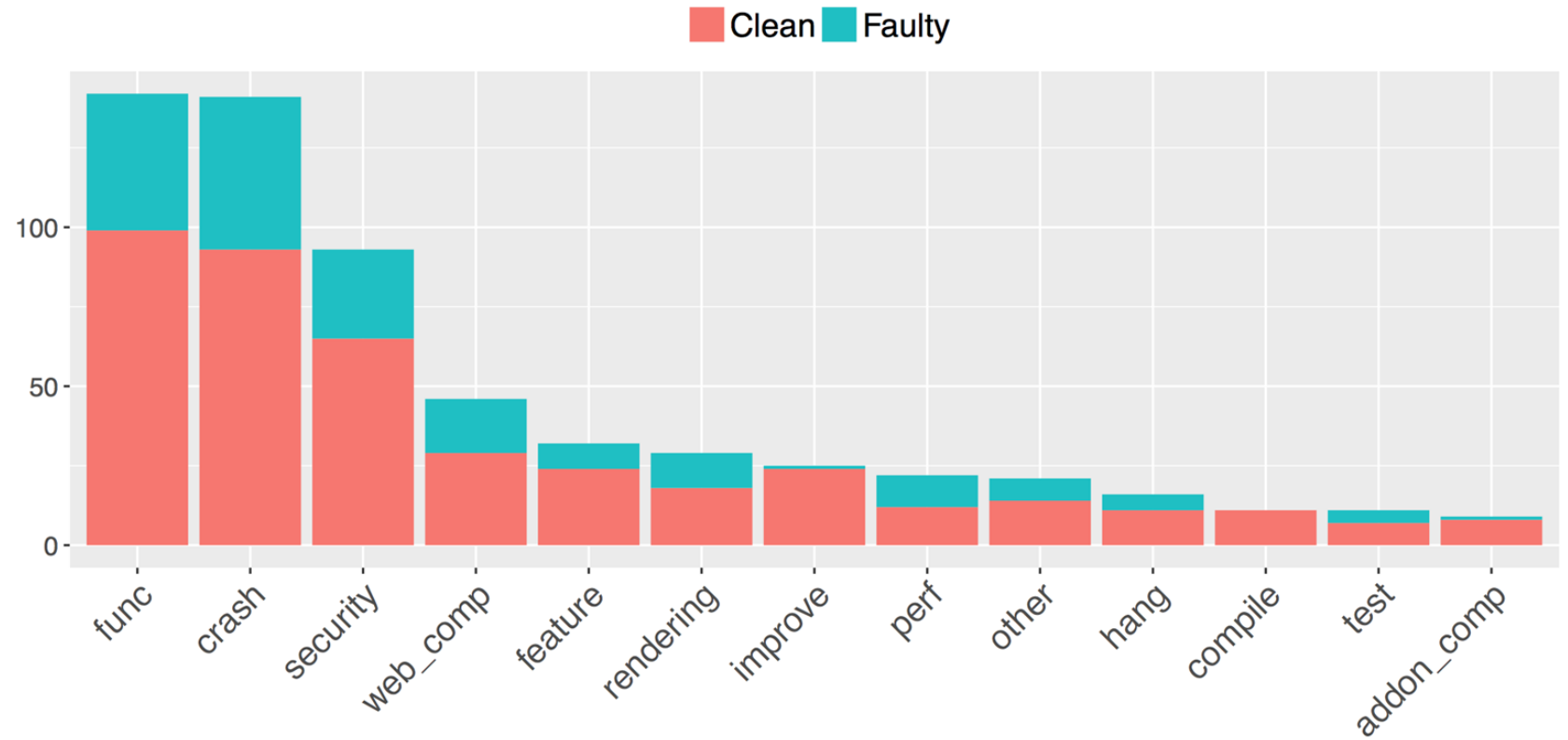
For all of the Aurora, Beta, and Release channels, uplifted patches have significant shorter response delta (with small effect size) than other patches.

Release managers' feedback: *“when I reject something, I won't make the call immediately. I will think about it before doing it, in case I change my mind or new facts are coming in the equation”.*

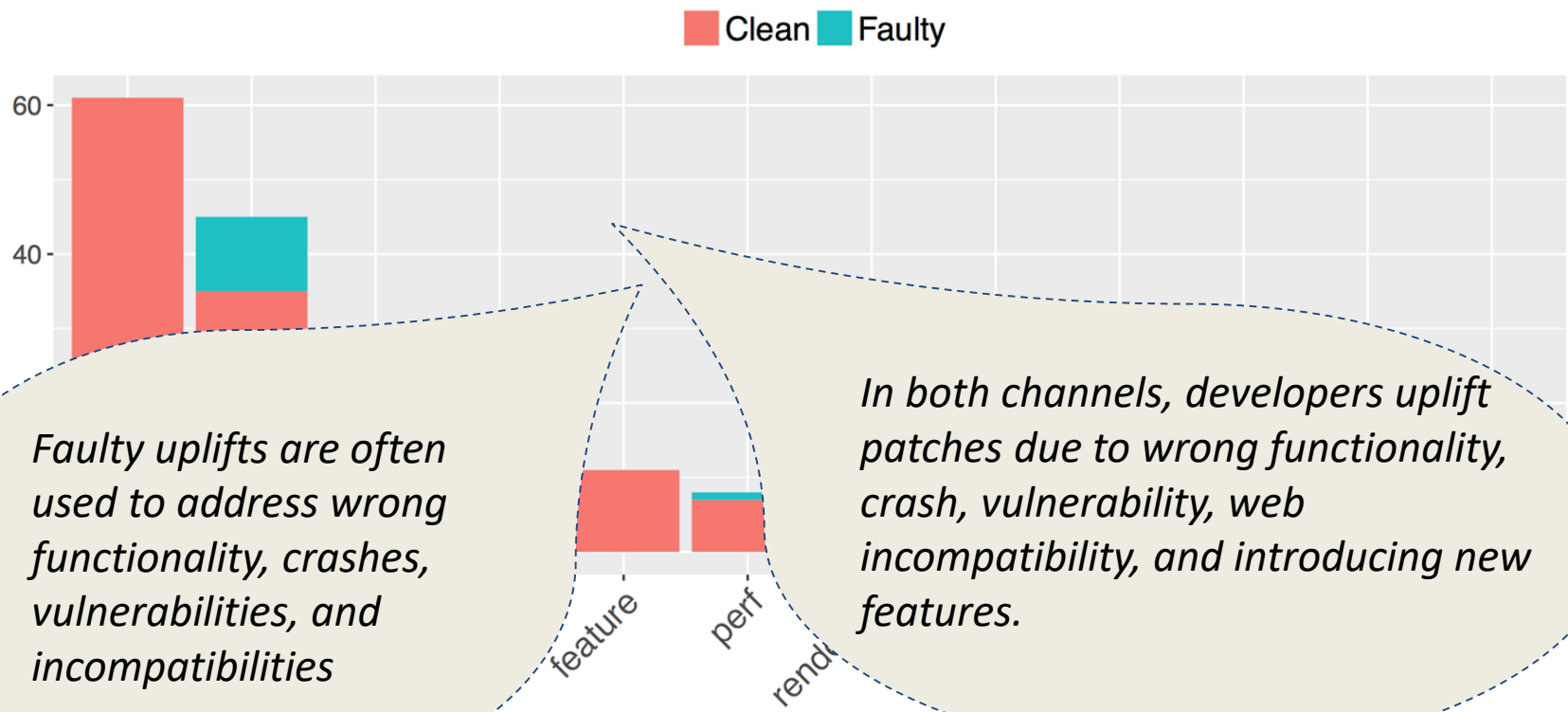
Other results are channel dependent.



# Results - RQ1 (uplift reason, Beta channel)



# Results - RQ1 (uplift reason, Release channel)



# Results - RQ1 (release manager interview)

“Which factors do you take into account when deciding about an uplift?”

Release managers said that they will consider *the importance of the issue, risk associated with the patch, timing of the uplift in stabilization cycles, and verification of the path.*

# Results - RQ1 (release manager interview)

“Are there differences in how you handle uplifts in different channels, and what are the differences?”

After the middle point of the Beta cycle, release managers only accept patches fixing high security issues, high volume crashes, severe recent regressions, severe performance issues or memory leaks.

# Results - RQ1 (release manager interview)

“How do you decide which developers you can trust?”

Release managers mentioned:

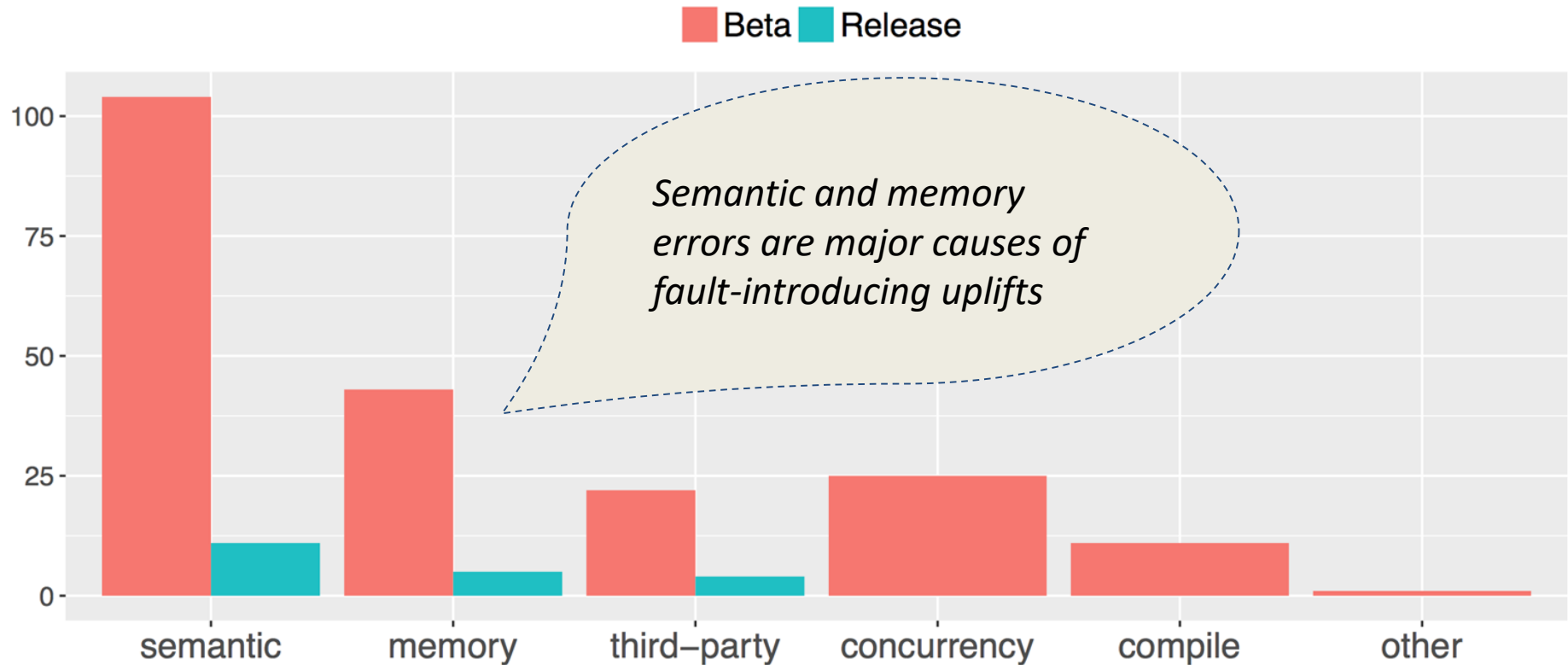
- *When they seem really overconfident or aren't telling me the whole story I lose some trust*
- *Some developers are taking a lot of risks, some other less and are super reactive to fix potential fallout*

# Results - RQ2 (statistical tests)

For all of the Aurora, Beta, and Release channels, fault-introducing uplifts have significantly larger patch size (with a small effect size) than clean uplifts.

Other results are channel dependent.

# Results - RQ2 (root causes of faulty uplifts)



# Results - RQ2 (release manager interview)

“What are the characteristics of the fault-introducing patches that you are not currently taking enough into account but could be considered in the future?”

All the release managers agreed that it would be beneficial for them to have more detailed information about the complexity of the targeted patches and more information about the history of the components involved in these patches.



# Conclusion

- Patch uplift allows to promote features or bug fixes directly from development channel to a stabilization channel.
- Patch uplift sometimes lead to faults.
- Software organizations can apply (or enhance their effort of using) static analysis tools to prevent memory-related faults.
- Reviewers and release managers should more carefully inspect uplifts that address: wrong functionality, crashes, vulnerabilities, and incompatibilities.
- Reviewers should make more effort on inspecting potential semantic faults.