Are Project Tests Enough for Automated Dependency Updates?

A Case Study of 262 Java Projects on Github

Joseph Hejderup

04-02-2024

ENDOR LABS

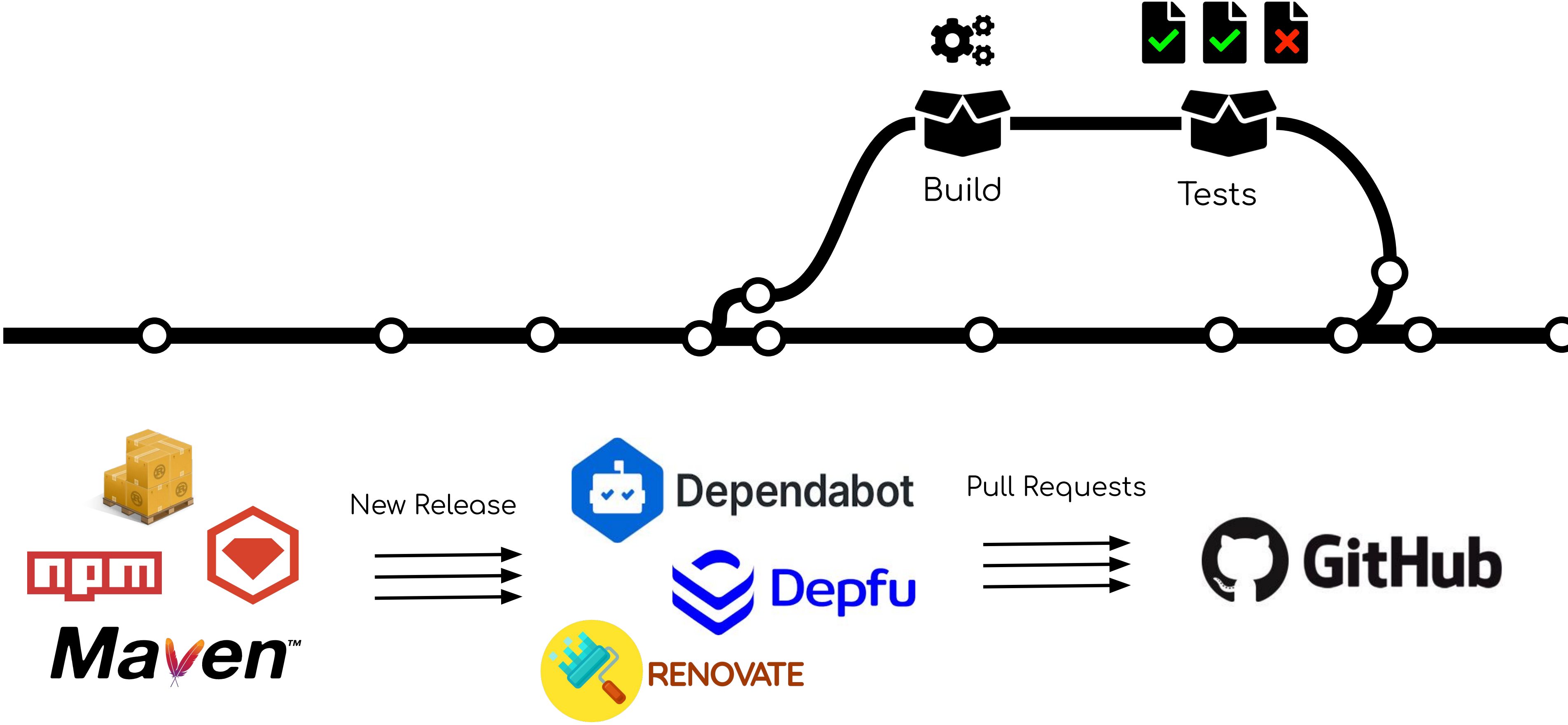
TUDelft

Member of Technical Staff, Endor Labs, Inc.
PhD Candidate, TU Delft, the Netherlands

Main Interests:

- Scaling Program Analysis
- Software Supply Chain Security

Automated Dependency Updates



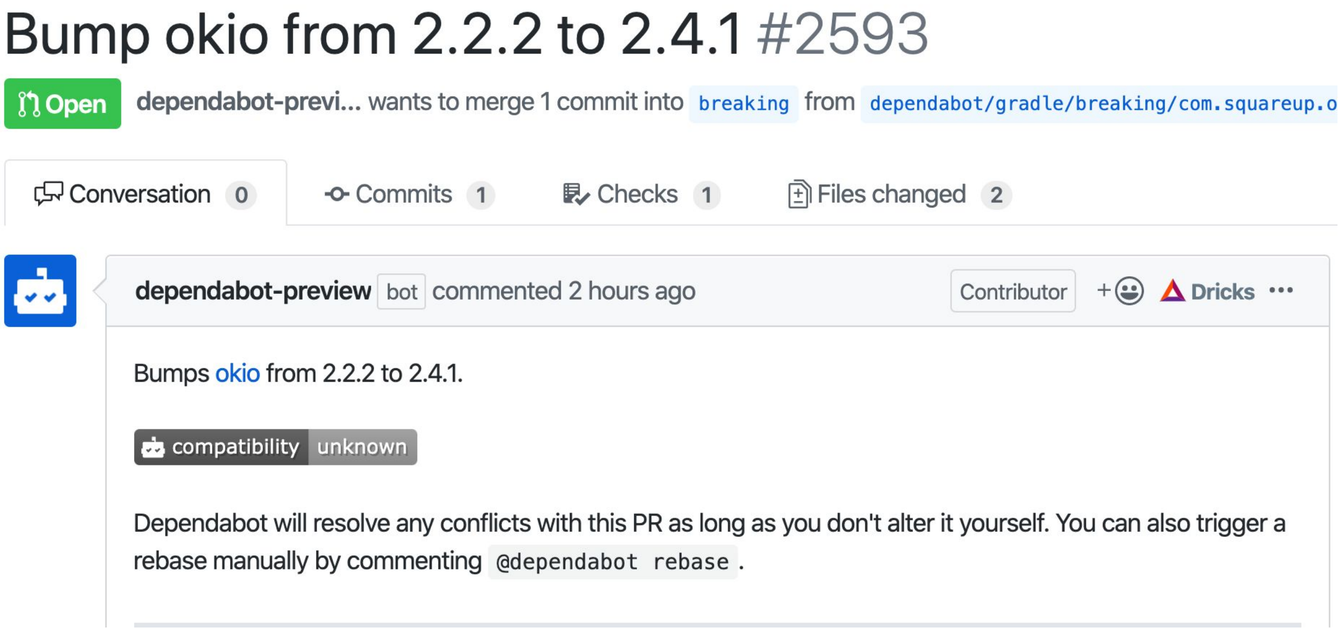
Build

Tests

New Release

Pull Requests

Automated Dependency Updates



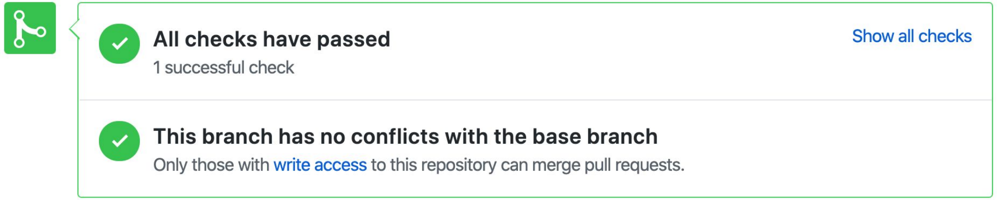
Bump okio from 2.2.2 to 2.4.1 #2593

dependabot-previ... wants to merge 1 commit into `breaking` from `dependabot/gradle/breaking/com.squareup.o`

dependabot-preview bot commented 2 hours ago

Bumps okio from 2.2.2 to 2.4.1.

Dependabot will resolve any conflicts with this PR as long as you don't alter it yourself. You can also trigger a rebase manually by commenting `@dependabot rebase`.



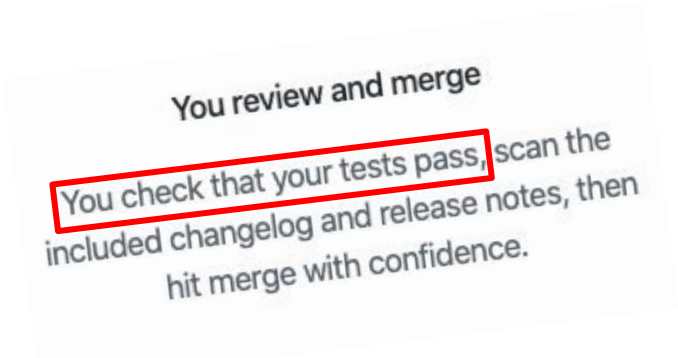
All checks have passed

1 successful check

This branch has no conflicts with the base branch

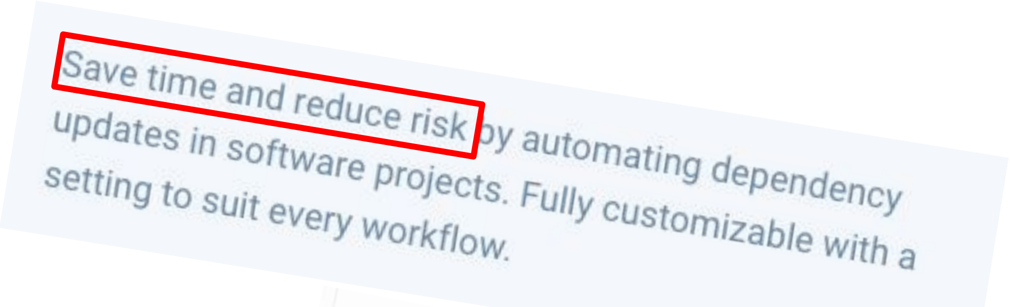
Only those with write access to this repository can merge pull requests.

Avoid Regressions?

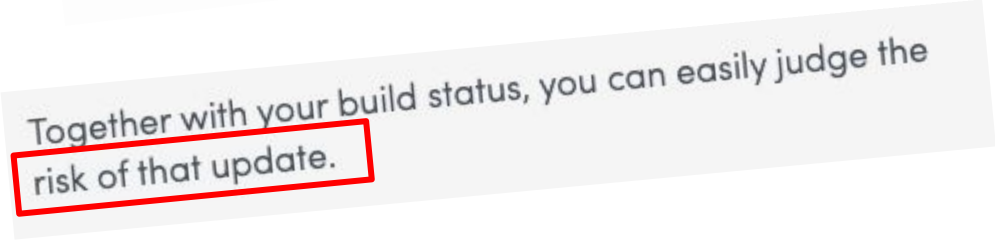


You review and merge

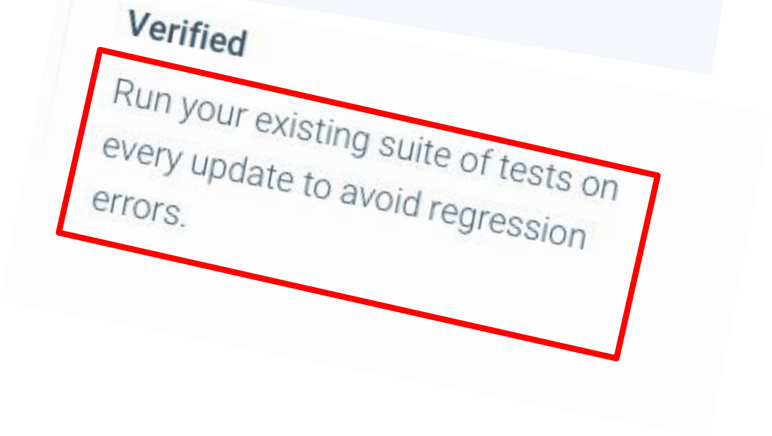
You check that your tests pass, scan the included changelog and release notes, then hit merge with confidence.



Save time and reduce risk by automating dependency updates in software projects. Fully customizable with a setting to suit every workflow.



Together with your build status, you can easily judge the risk of that update.



Verified

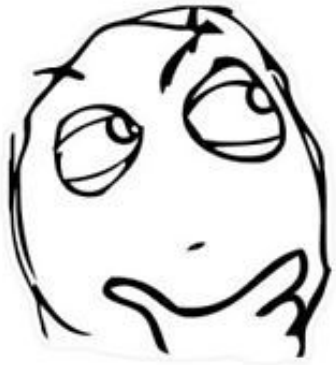
Run your existing suite of tests on every update to avoid regression errors.

Test Suites + Third-Party Libraries

1. Do we even write tests against dependencies in the first place?

2. Do project test suites even cover usages of dependencies in the source code?

3. Are tests sufficient alone for detecting bad updates?



Q: Should we write tests for dependencies/third-party libraries?

Empirical Study

- What is the **statement coverage** of **function calls** to dependencies?

- How **effective** are test suites in detecting updates with **regression errors**?

- How does **static analysis complement/compare** to test suites in updating dependencies?

Statement Coverage: How?

Direct & Transitive Dependencies

❑ **Direct Dependencies:** Extract call sites of third-party libs in bytecode

❑ **Transitive Dependencies:** Static Call Graph to infer call paths to transitive call sites

❑ **Instrumentation**: Instrument functions belonging to dependencies and record their execution

WALA

T. J. WATSON LIBRARIES FOR ANALYSIS

asm/asm

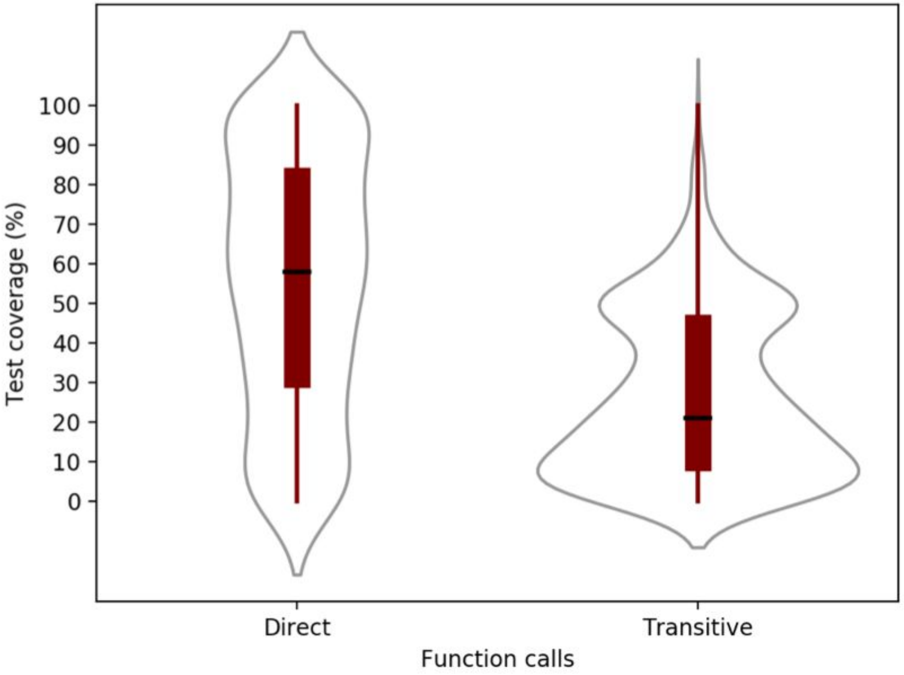
wala/WALA

Statement Coverage

521 GH Projects having tests

60%

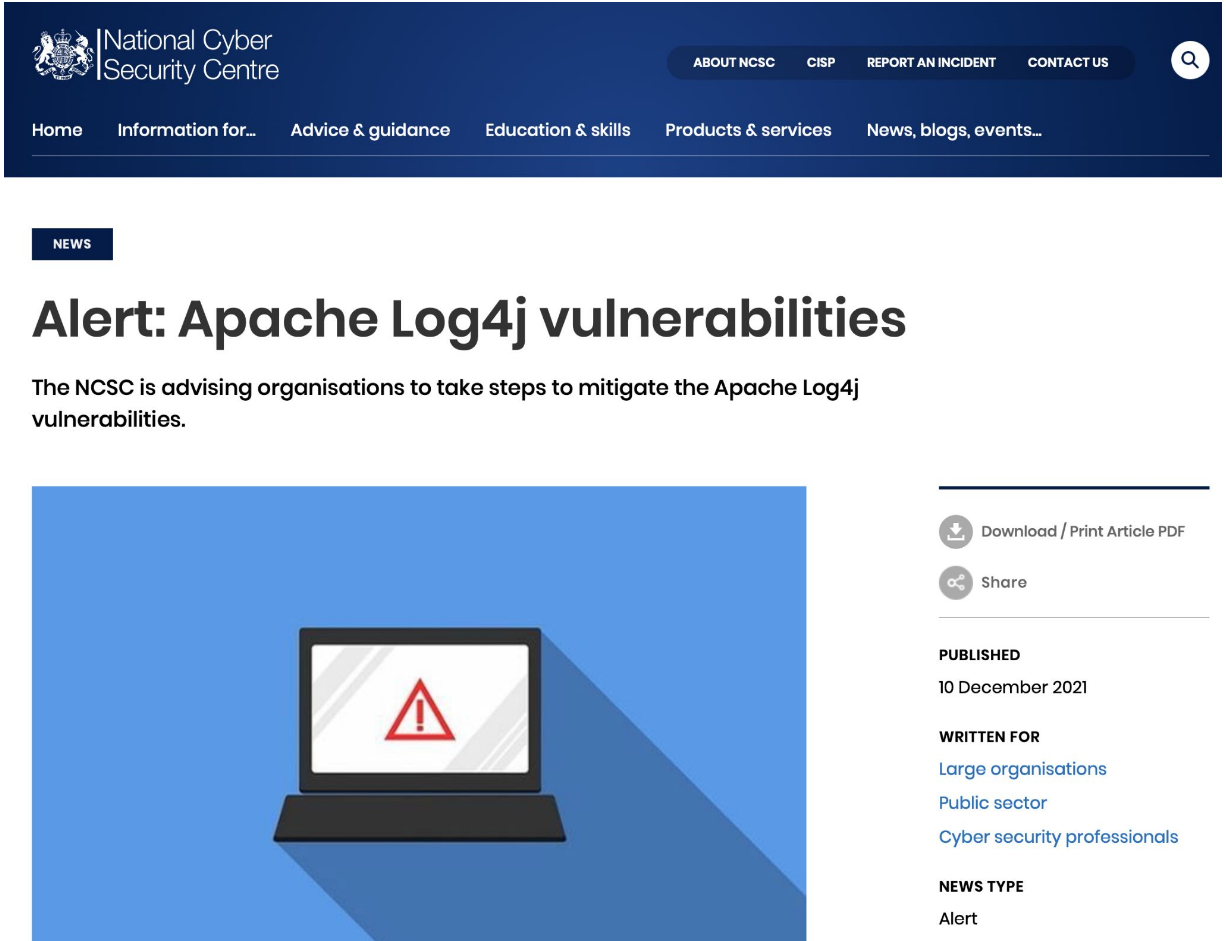
median coverage of **direct dependencies**



20% median coverage of **transitive dependencies** ⚠

Updates on untested code!

Does this matter at all?



Alert: Apache Log4j vulnerabilities

The NCSC is advising organisations to take steps to mitigate the Apache Log4j vulnerabilities.

PUBLISHED

10 December 2021

WRITTEN FOR

Large organisations

Public sector

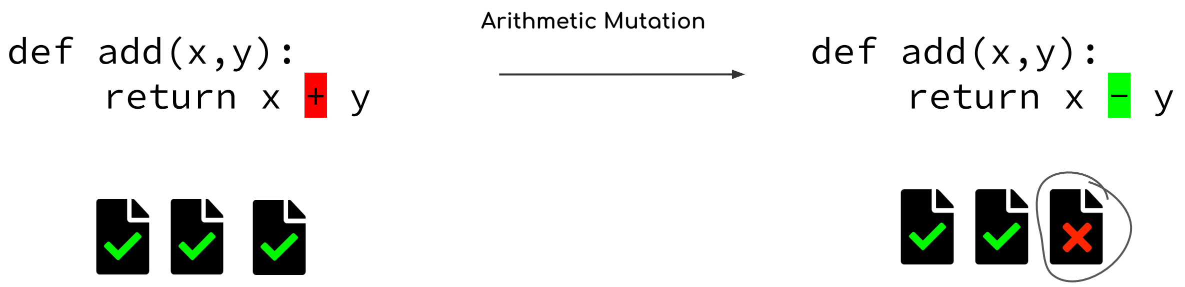
Cyber security professionals

NEWS TYPE

Alert

Test Effectiveness: How?

Mutation testing!



```
def add(x,y):
    return x + y
```

✓ ✓ ✓

Arithmetic Mutation

```
def add(x,y):
    return x - y
```

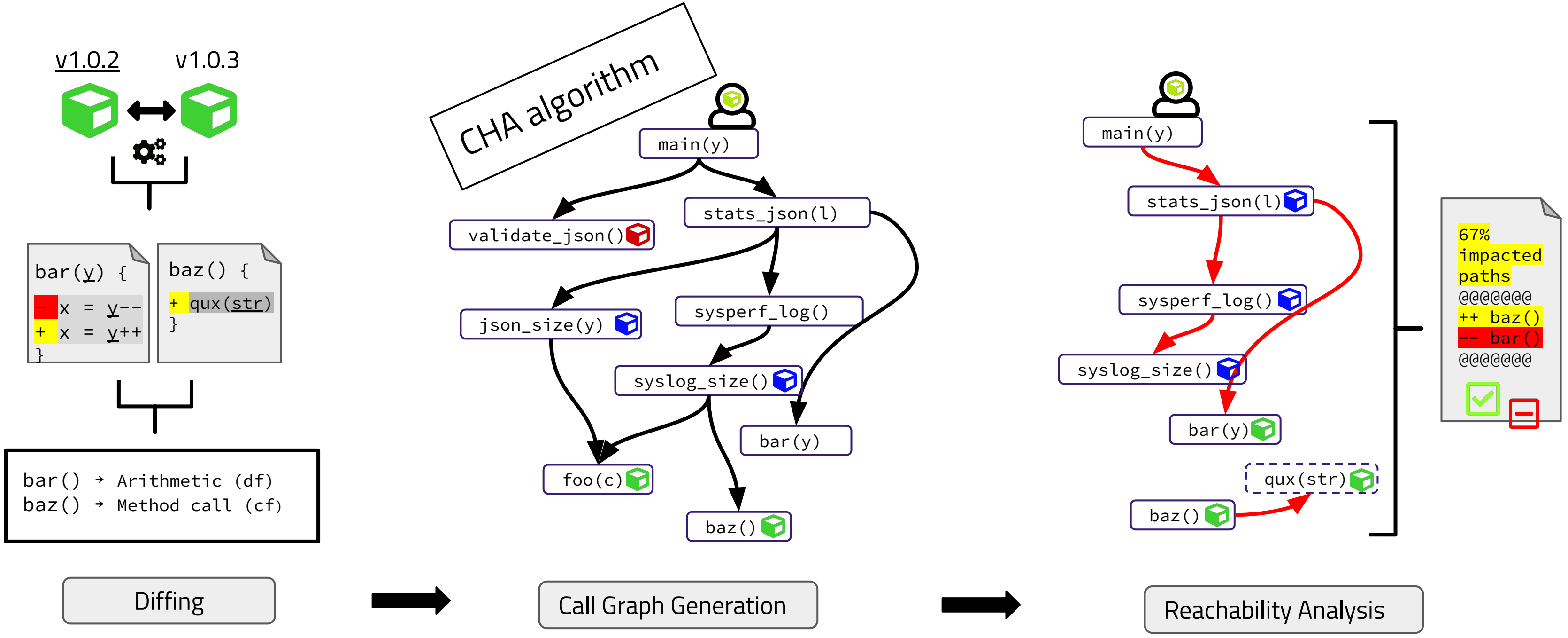
✓ ✓ ✗

pitest.org

We use PITest with a twist: We don’t mutate all dependency functions; only those reachable by tests!

Uppdatera

Change Impact Analysis as an alternative!



v1.0.2 v1.0.3

```
bar(y) {
-  x = y--
+  x = y++
}
```

```
baz() {
+ qux(str)
}
```

```
bar() → Arithmetic (df)
baz() → Method call (cf)
```

Diffing

CHA algorithm

main(y)

validate_json()

stats_json(l)

json_size(y)

sysperf_log()

syslog_size()

bar(y)

foo(c)

baz()

Call Graph Generation

main(y)

stats_json(l)

sysperf_log()

syslog_size()

bar(y)

qux(str)

baz()

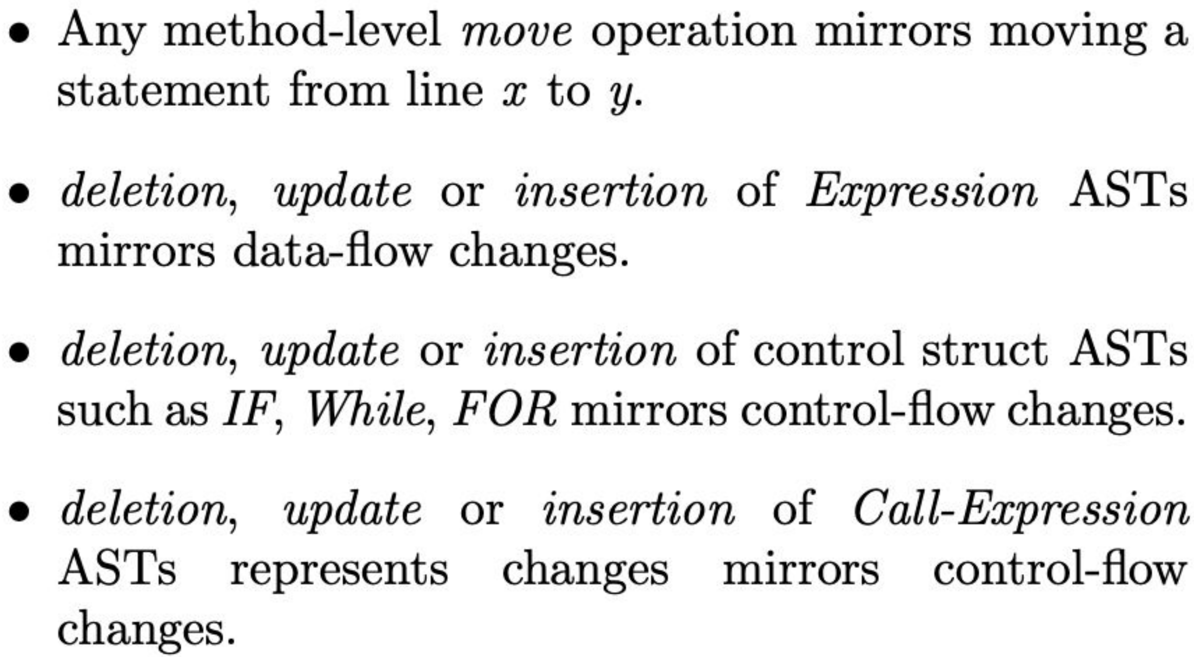
```
67%
impacted
paths
@@@@@@@
++ baz()
-- bar()
@@@@@@@
```

Reachability Analysis

jhejderup/Uppdatera

How to deal with Semantic Changes?

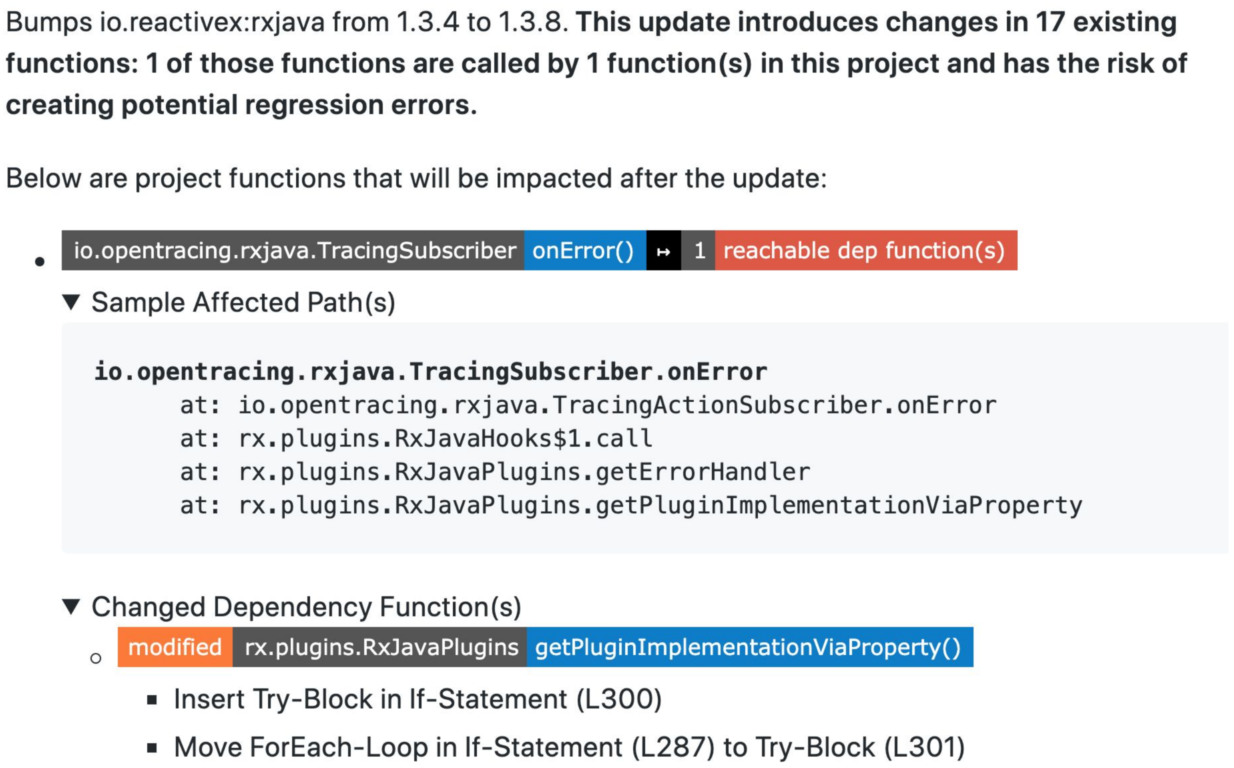
Behavioural Changes: Data-flow and Control-flow changes!



- Any method-level *move* operation mirrors moving a statement from line x to y.
- *deletion*, *update* or *insertion* of *Expression* ASTs mirrors data-flow changes.
- *deletion*, *update* or *insertion* of control struct ASTs such as *IF*, *While*, *FOR* mirrors control-flow changes.
- *deletion*, *update* or *insertion* of *Call-Expression* ASTs represents changes mirrors control-flow changes.

Uppdatera

Change Impact Analysis as an alternative!



Bumps io.reactivex:rxjava from 1.3.4 to 1.3.8. **This update introduces changes in 17 existing functions: 1 of those functions are called by 1 function(s) in this project and has the risk of creating potential regression errors.**

Below are project functions that will be impacted after the update:

- io.opentracing.rxjava.TracingSubscriber onError() ↦ 1 reachable dep function(s)

  ▼ Sample Affected Path(s)

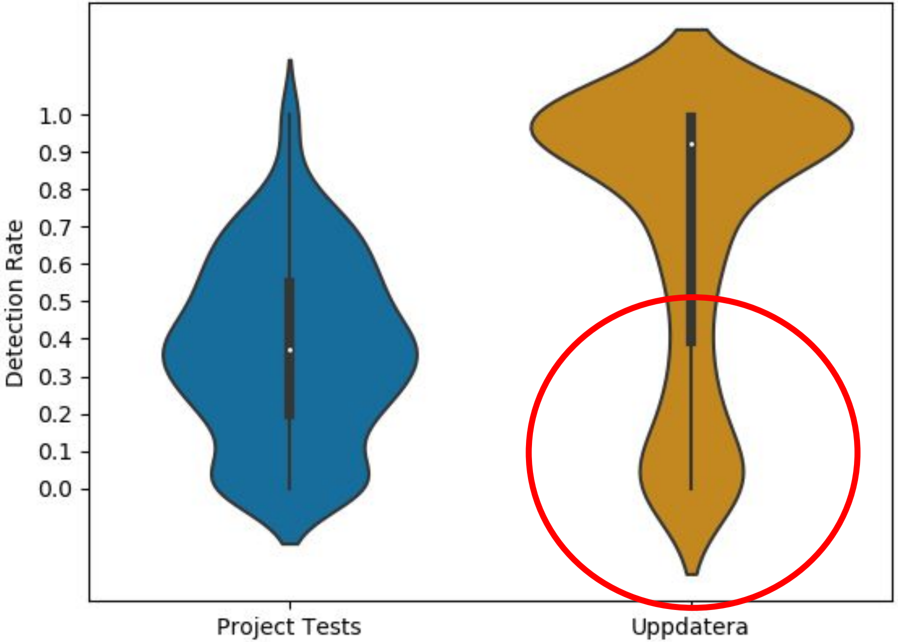
  ```
  io.opentracing.rxjava.TracingSubscriber.onError
          at: io.opentracing.rxjava.TracingActionSubscriber.onError
          at: rx.plugins.RxJavaHooks$1.call
          at: rx.plugins.RxJavaPlugins.getErrorHandler
          at: rx.plugins.RxJavaPlugins.getPluginImplementationViaProperty
  ```

  ▼ Changed Dependency Function(s)

  - modified rx.plugins.RxJavaPlugins getPluginImplementationViaProperty()
    - Insert Try-Block in If-Statement (L300)
    - Move ForEach-Loop in If-Statement (L287) to Try-Block (L301)

Test Effectivies

1 Million artificial updates on 262 GH Projects



On average,

37% detected by tests!

72% detected by Uppdatera!

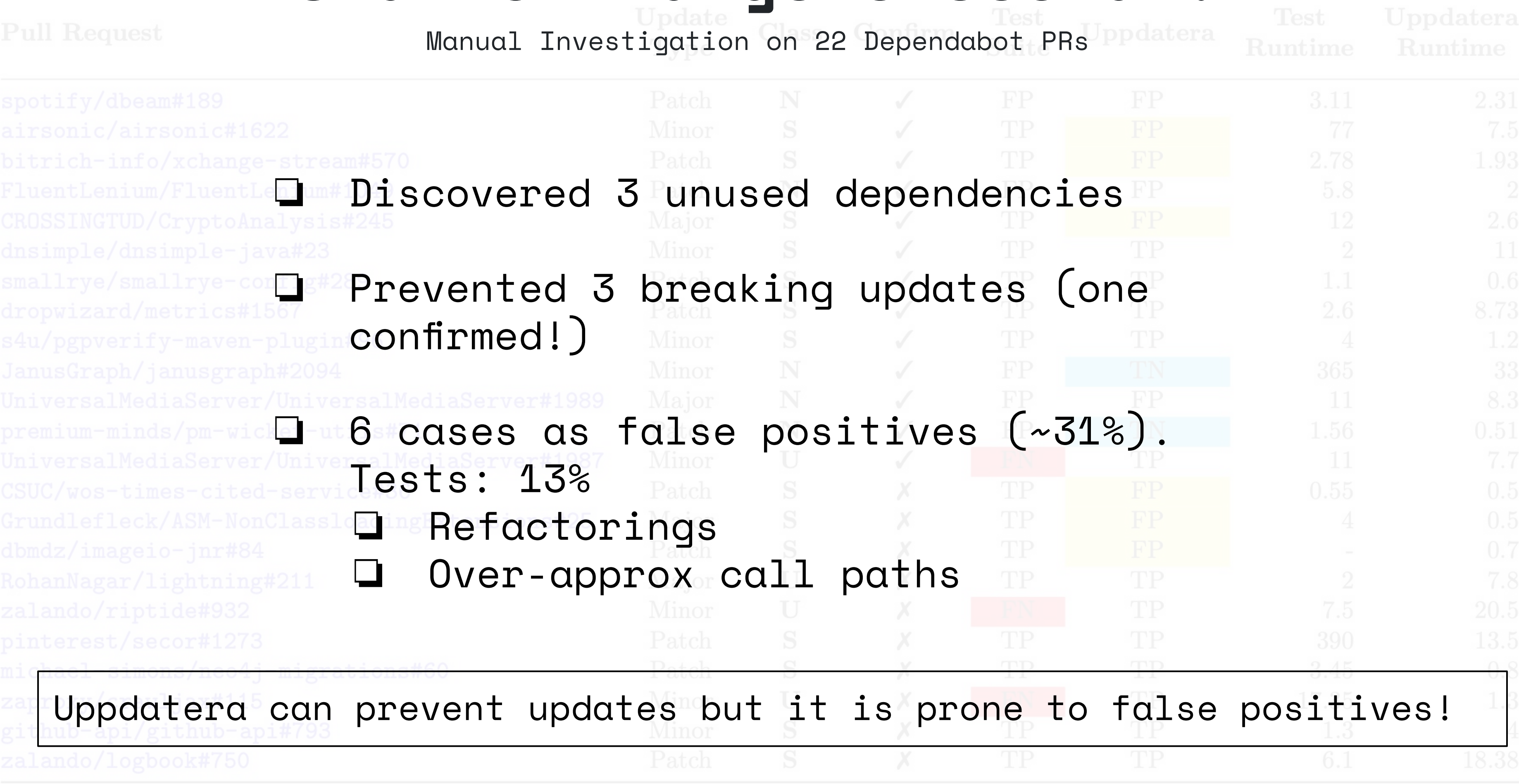
No guarantees that tests can prevent bad updates!

Static Analysis Useful?

Manual Investigation on 22 Dependabot PRs

- ❏ Discovered 3 unused dependencies
- ❏ Prevented 3 breaking updates (one confirmed!)
- ❏ 6 cases as false positives (~31%). Tests: 13%
  - ❏ Refactorings
  - ❏ Over-approx call paths

Uppdatera can prevent updates but it is prone to false positives!



Pull Request	Update Type	Class	Confirm	Test Suite	Uppdatera	Test Runtime	Uppdatera Runtime
spotify/dbeam#189	Patch	N	✓	FP	FP	3.11	2.31
airsonic/airsonic#1622	Minor	S	✓	TP	FP	77	7.5
bitrich-info/xchange-stream#570	Patch	S	✓	TP	FP	2.78	1.93
FluentLenium/FluentLenium#1[illegible]	Patch	N	✓	FP	FP	5.8	2
CROSSINGTUD/CryptoAnalysis#245	Major	S	✓	TP	FP	12	2.6
dnsimple/dnsimple-java#23	Minor	S	✓	TP	TP	2	11
smallrye/smallrye-config#28[illegible]	Patch	S	✓	TP	TP	1.1	0.6
dropwizard/metrics#1567	Patch	S	✓	TP	TP	2.6	8.73
s4u/pgpverify-maven-plugin#[illegible]	Minor	S	✓	TP	TP	4	1.2
JanusGraph/janusgraph#2094	Minor	N	✓	FP	TN	365	33
UniversalMediaServer/UniversalMediaServer#1989	Major	N	✓	FP	FP	11	8.3
premium-minds/pm-wicket-utils#[illegible]	Patch	N	✓	FP	TN	1.56	0.51
UniversalMediaServer/UniversalMediaServer#1987	Minor	U	✓	FN	TP	11	7.7
CSUC/wos-times-cited-service#80	Patch	S	✗	TP	FP	0.55	0.5
Grundlefleck/ASM-NonClassloadingExtractors#25	Major	S	✗	TP	FP	4	0.5
dbmdz/imageio-jnr#84	Patch	S	✗	TP	FP	-	0.7
RohanNagar/lightning#211	Major	U	✗	TP	TP	2	7.8
zalando/riptide#932	Minor	U	✗	FN	TP	7.5	20.5
pinterest/secor#1273	Patch	S	✗	TP	TP	390	13.5
michael-simons/neo4j-migrations#60	Patch	S	✗	TP	TP	3.45	0.8
zap[illegible]#115	Minor	U	✗	FN	TP	[illegible]	1.3
github-api/github-api#793	Minor	S	✗	TP	TP	1.3	[illegible]4
zalando/logbook#750	Patch	S	✗	TP	TP	6.1	18.38

Recommendations

Tool Makers

- ❑ Confidence Score
  - ❑ How reliable is my test suite for a particular library?
  - ❑ Indication on where to direct test efforts

- ❑ Gaps in Test Coverage
  - ❑ Complement with Static Analysis
  - ❑ Catch early errors without running build/tests

Recommendations

Users of Automated Updating

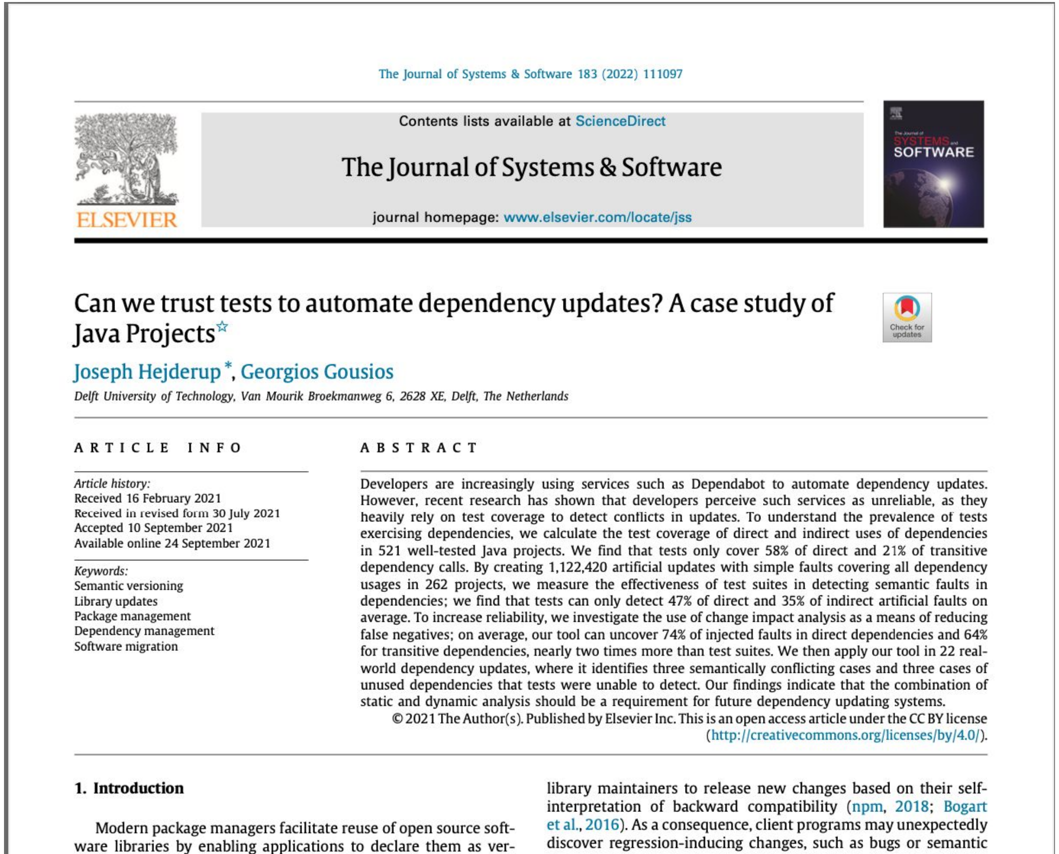
- ❏ Reuse is “free” but the operational/maintenance costs are not “free”

- ❏ Should not blindly trust automated dependency updates—I guess no one does this :D

- ❏ Write tests for critical dependencies

Want to know more?

https://doi.org/10.1016/j.jss.2021.111097 (Open Access)



The Journal of Systems & Software 183 (2022) 111097

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Can we trust tests to automate dependency updates? A case study of Java Projects☆

Joseph Hejderup*, Georgios Gousios

Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 16 February 2021
Received in revised form 30 July 2021
Accepted 10 September 2021
Available online 24 September 2021

Keywords:
Semantic versioning
Library updates
Package management
Dependency management
Software migration

ABSTRACT

Developers are increasingly using services such as Dependabot to automate dependency updates. However, recent research has shown that developers perceive such services as unreliable, as they heavily rely on test coverage to detect conflicts in updates. To understand the prevalence of tests exercising dependencies, we calculate the test coverage of direct and indirect uses of dependencies in 521 well-tested Java projects. We find that tests only cover 58% of direct and 21% of transitive dependency calls. By creating 1,122,420 artificial updates with simple faults covering all dependency usages in 262 projects, we measure the effectiveness of test suites in detecting semantic faults in dependencies; we find that tests can only detect 47% of direct and 35% of indirect artificial faults on average. To increase reliability, we investigate the use of change impact analysis as a means of reducing false negatives; on average, our tool can uncover 74% of injected faults in direct dependencies and 64% for transitive dependencies, nearly two times more than test suites. We then apply our tool in 22 real-world dependency updates, where it identifies three semantically conflicting cases and three cases of unused dependencies that tests were unable to detect. Our findings indicate that the combination of static and dynamic analysis should be a requirement for future dependency updating systems.

1. Introduction

Modern package managers facilitate reuse of open source software libraries by enabling applications to declare them as ver-

library maintainers to release new changes based on their self-interpretation of backward compatibility (npm, 2018; Bogart et al., 2016). As a consequence, client programs may unexpectedly discover regression-inducing changes, such as bugs or semantic