Are Project Tests Enough for Automated Dependency Updates?

A Case Study of 262 Java Projects on Github

Joseph Hejderup
04-02-2024
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

Bump okio from 2.2.2 to 2.4.1 #2593

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.

Verified

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

Together with your build status, you can easily judge the risk of that update.
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
60% median coverage of direct dependencies

20% median coverage of transitive dependencies

Updates on untested code!
Alert: Apache Log4j vulnerabilities

The NCSC is advising organisations to take steps to mitigate the Apache Log4j vulnerabilities.
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!
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.TracingAction Subscriber.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 Effectivities

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!
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!
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
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
Revised 18 April 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 highly rely on test coverage to detect conflicts in updates. To understand the presence of tests exercising dependencies, we calculate the test coverage of direct and indirect uses of dependencies in 321 well-tested Java projects. We find that tests only cover 58% of direct and 21% of transitive dependency calls. By creating 1,122,400 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.

© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Modern package managers facilitate reuse of open source software libraries by enabling applications to declare them as version-independent dependencies. Library maintainers 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 differences.