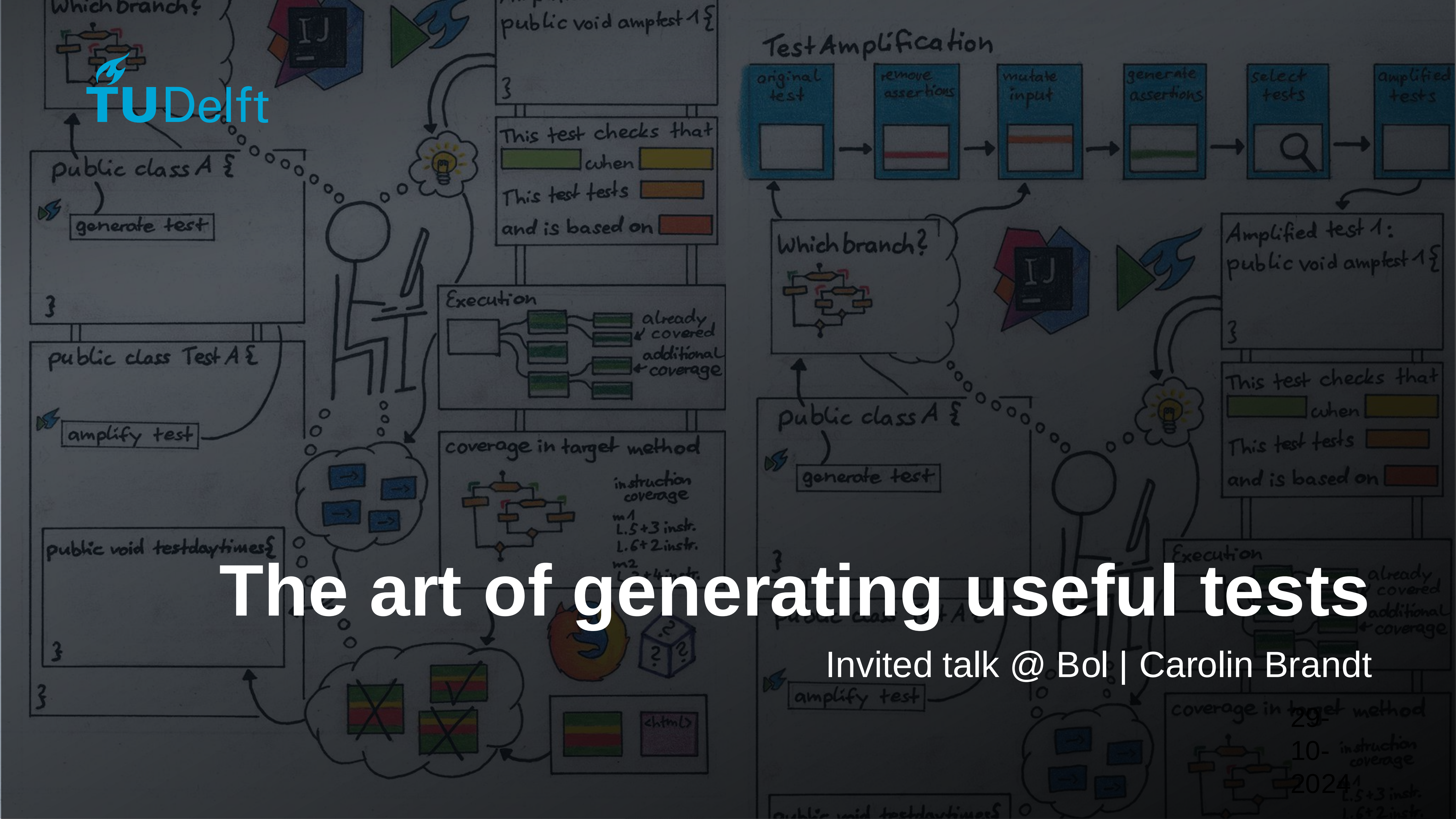
TU Delft



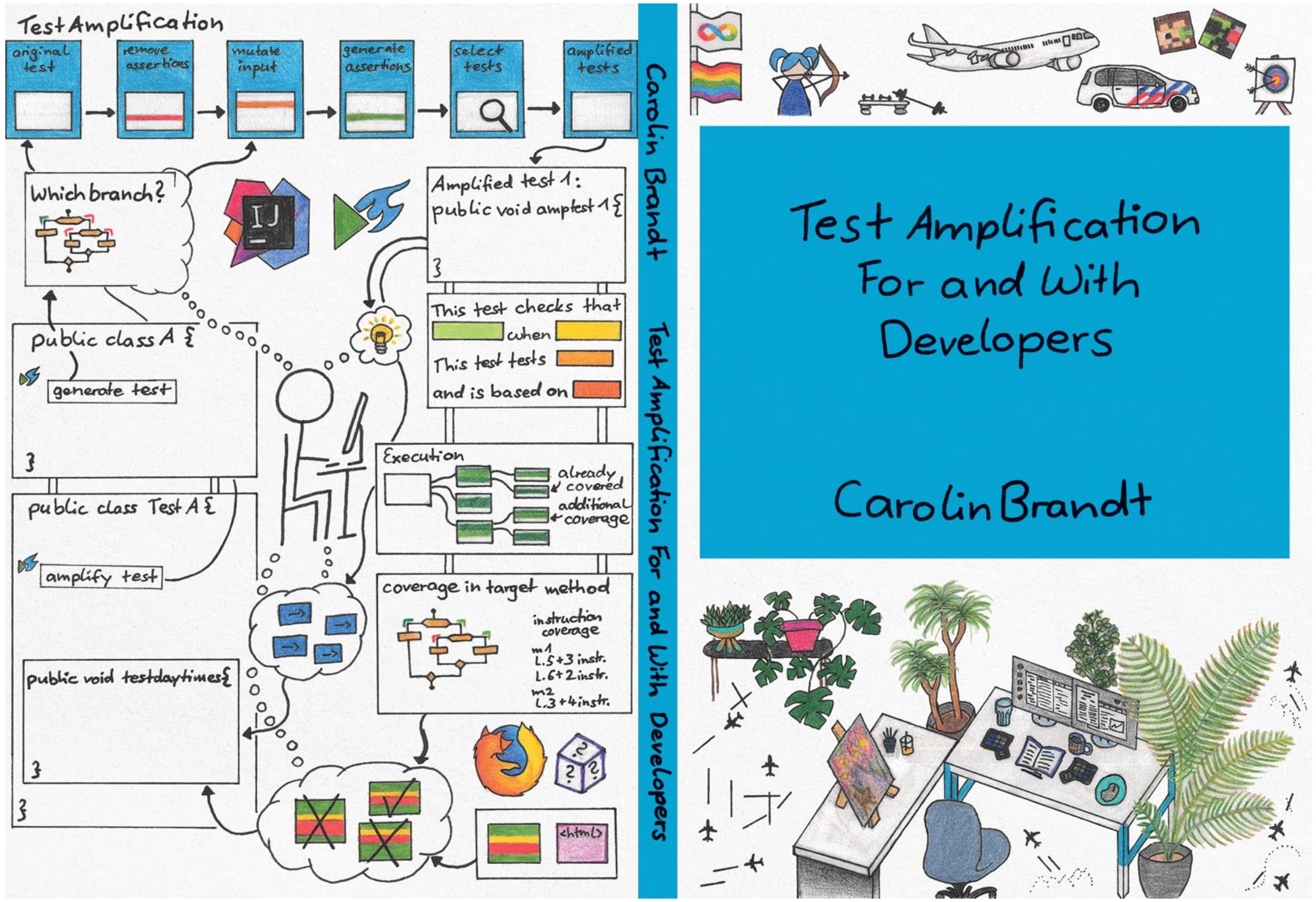
The art of generating useful tests

Invited talk @ Bol | Carolin Brandt

29-10-2024

Introduction:
Carolin Brandt

- Assistant Professor @ TU Delft
- PhD in Software Engineering
- Focus:
  Developer-centric approach for software engineering tools and automations



What is automatic test generation?

What are tests?

- Automated / programmed xUnit or integration tests

Developer Testing

- Manual tests / end-to-end tests
  - Can be defined in natural language

QA Testing

- Reliability / Load / Security testing
  - Crash reproduction

“Non-Functional Aspect" Testing

Normally, all of these require engineers to manually write test scripts….

Software testing is …

Slow

Painful

Boring

Necessary

Automation to the rescue !

A plethora of ways to generate tests

We built many ways to generate tests…

Search-based / Genetic Algorithms

Fuzzing

Random Testing

Test Carving

Test Amplification

AI / ML-based

Hybrid

Model Checking

Symbolic Execution

TU Delft

We built many ways to generate tests…

Generate Test Data

Generate Test Code

Fuzzing

Random Testing

Search-based / Genetic Algorithms

AI / ML-based

Symbolic Execution

Test Carving

Model Checking

Hybrid

Test Amplification

What tools are available for test generation?

Test Generation Tools

Quite dominated by research tools….

- **TestSpark IntelliJ Plugin** (https://plugins.jetbrains.com/plugin/21024-testspark, https://github.com/JetBrains-Research/TestSpark): Java and Kotlin, LLM (openAI, internal JetBrains AI) & Search-based (local, EvoSuite)
- **GitHub Test Pilot** (https://githubnext.com/projects/testpilot/): JavaScript / TypeScript, LLM (GitHub Copilot)
- **TestCube IntelliJ Plugin** (https://plugins.jetbrains.com/plugin/14678-test-cube): Java, Test Amplification (local, DSpot)

Lots of detailed / CLI tools:

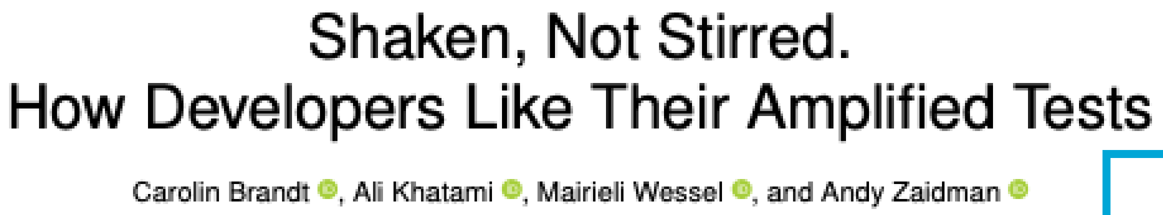
EvoSuite (search-based, Java), Pyguin (search-based, python), DSpot (test amplification, Java),
Plenty Fuzzing tools (https://github.com/secfigo/Awesome-Fuzzing)

A dive into my recent research:

Developer-experience / involvement with generated tests

TU Delft

Developer-Experience with Generated Tests

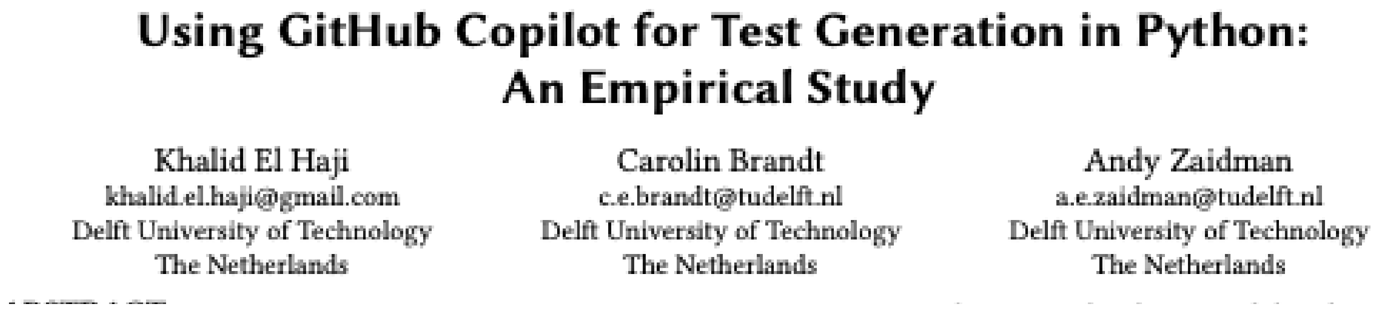


Shaken, Not Stirred.
How Developers Like Their Amplified Tests

Carolin Brandt, Ali Khatami, Mairieli Wessel, and Andy Zaidman

IEEE Transactions on
Software Engineering, 2024

Amplified tests + open-source maintainers: What do developers change in generated tests before adding them into their test suite?



Using GitHub Copilot for Test Generation in Python:
An Empirical Study

Khalid El Haji	Carolin Brandt	Andy Zaidman
khalid.el.haji@gmail.com	c.e.brandt@tudelft.nl	a.e.zaidman@tudelft.nl
Delft University of Technology	Delft University of Technology	Delft University of Technology
The Netherlands	The Netherlands	The Netherlands

International Conference on
Automation of Software Test 2024

GH Copilot for test generation: How well does Copilot work to generate tests out of the box?

TU Delft

Shaken, Not Stirred.

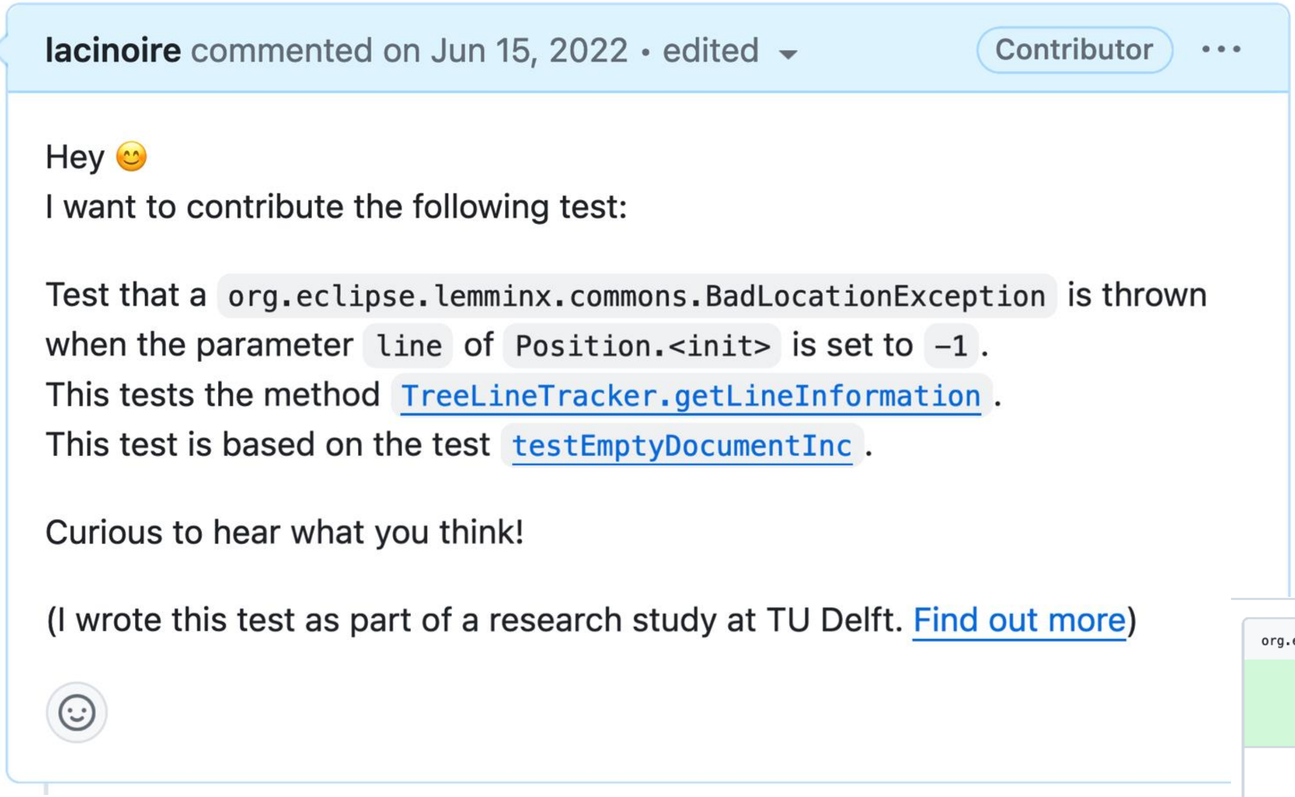
How Developers Like Their Amplified Tests

Carolin Brandt, Ali Khatami, Mairieli Wessel, Andy Zaidman

Delft University of Technology, Radboud University

IEEE Transactions on Software Engineering 2024

Open Source Contribution Study

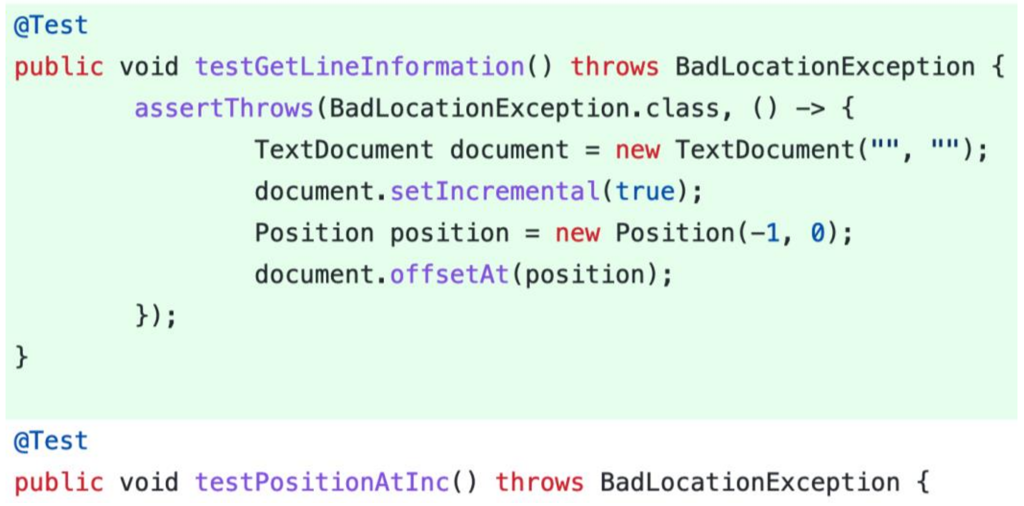

lacinoire commented on Jun 15, 2022 • edited

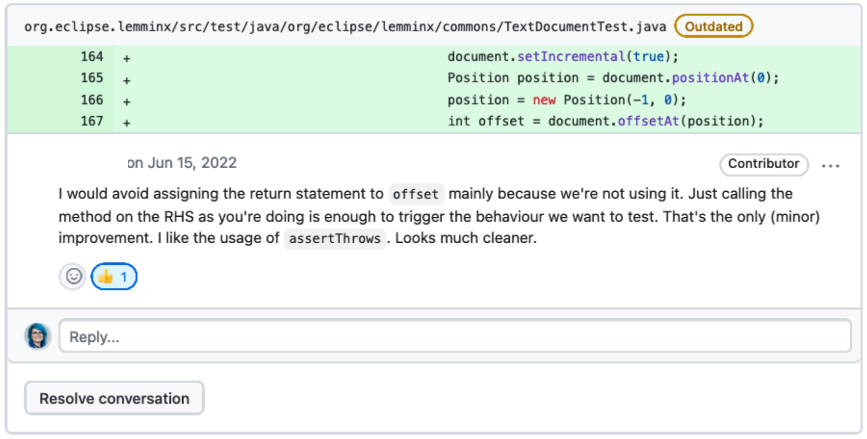
Hey 😊
I want to contribute the following test:

Test that a `org.eclipse.lemminx.commons.BadLocationException` is thrown when the parameter `line` of `Position.<init>` is set to `-1`.
This tests the method `TreeLineTracker.getLineInformation`.
This test is based on the test `testEmptyDocumentInc`.

Curious to hear what you think!

(I wrote this test as part of a research study at TU Delft. Find out more)


```java
@Test
public void testGetLineInformation() throws BadLocationException {
        assertThrows(BadLocationException.class, () -> {
                TextDocument document = new TextDocument("", "");
                document.setIncremental(true);
                Position position = new Position(-1, 0);
                document.offsetAt(position);
        });
}

@Test
public void testPositionAtInc() throws BadLocationException {
```


org.eclipse.lemminx/src/test/java/org/eclipse/lemminx/commons/TextDocumentTest.java

```java
+                document.setIncremental(true);
+                Position position = document.positionAt(0);
+                position = new Position(-1, 0);
+                int offset = document.offsetAt(position);
```

on Jun 15, 2022

I would avoid assigning the return statement to `offset` mainly because we're not using it. Just calling the method on the RHS as you're doing is enough to trigger the behaviour we want to test. That's the only (minor) improvement. I like the usage of `assertThrows`. Looks much cleaner.

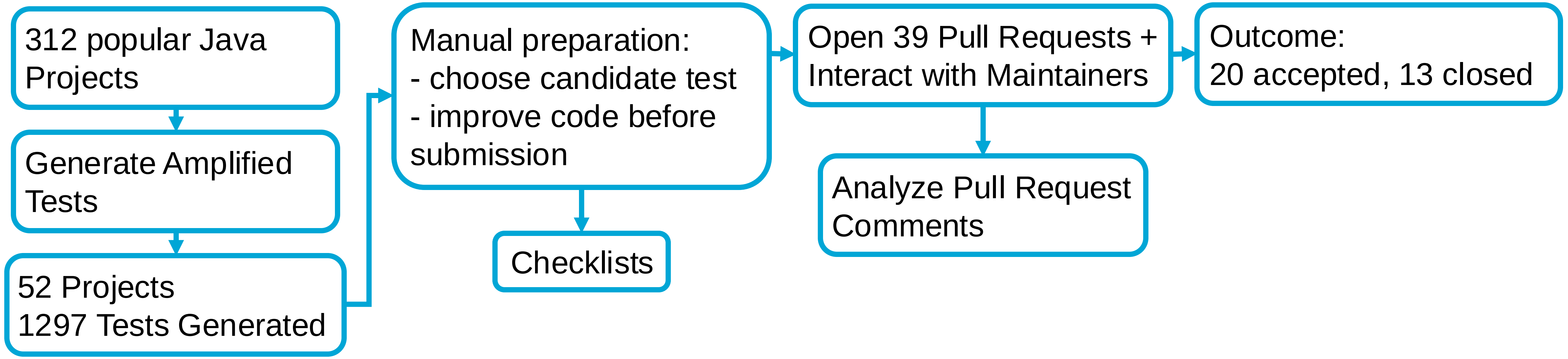
Not all tests can be submitted as is…

```
/**
* Test that ((tech.tablesaw.api.Table) (summary.dropRowsWithMissingValues())).toString() is equal to " Is false \n Value | Count |\n-------------------\n false | 2 |\n true | 5 |"
when dropRowsWithMissingValues() is called.
This tests the methods NumberColumn.getPrintFormatter and NumberColumn.emptyCopy and NumberColumnFormatter.format and AbstractStringColumn.set and
AbstractStringColumn.getString and StringColumn.isMissing and StringColumn.appendMissing and StringColumn.emptyCopy and StringColumn.set and Table.name and
Table.checkColumnSize and Table.emptyCopy and Table.dropRowsWithMissingValues and NumericColumn.isEmpty and StringColumnFormatter.format and
DataFramePrinter.lambda$print$5 and DataFramePrinter.lambda$whitespace$4 and DataFramePrinter.whitespace and DataFramePrinter.lambda$getHeaderTokens$6
and DataFramePrinter.<init> and DataFramePrinter.getDataToken and DataFramePrinter.lambda$getDataTemplate$3 and
DataFramePrinter.lambda$getHeaderTemplate$1 and DataFramePrinter.getDataTemplate and DataFramePrinter.getHeaderTemplate and
DataFramePrinter.getHeaderTokens and DataFramePrinter.lambda$getDataTemplate$2 and DataFramePrinter.tableName and
DataFramePrinter.lambda$getHeaderTemplate$0 and DataFramePrinter.getDataTokens and DataFramePrinter.getWidths and DataFramePrinter.print and
BitmapBackedSelection.toBitmap and BitmapBackedSelection.andNot and StringUtils.repeat and ByteDictionaryMap.appendMissing and
ByteDictionaryMap.getKeyForIndex and ByteDictionaryMap.isMissing and ByteDictionaryMap.append and ByteDictionaryMap.set and DoubleColumn.isMissingValue and
DoubleColumn.isMissing and DoubleColumn.getString and DoubleColumn.set and AbstractColumnType.equals and Relation.isEmpty and Relation.toString and Relation.print.
The test is based on testBitmapConstructor.
*/
@Test
  public void testNumberColumn.emptyCopyAndAbstractStringColumn.set() throws Exception {
  BooleanColumn bc = BooleanColumn.create("Is false", column.isFalse(), column.size());
  Table summary = bc.summary(); Assertions.assertEquals(" Is false \n Value | Count |\n-------------------\n false | 2 |\n true | 5 |",
summary.dropRowsWithMissingValues().toString());
}
```

Not all tests can be submitted as is…

```
/**
* Test that simpleAuthId is equal to "testSimple" when setSocialContent(...) is called with the parameter socialContent = "QQmcN(DJ9-f?bfZ`LvH&". This tests the method
AuthorizableConfigBean.setSocialContent. The test is based on testCollectPrincipalsToBeMigrated.
*/
@Test(timeout = 10000)
public void testAuthorizableConfigBean.setSocialContent() throws Exception {
AuthorizablesConfig authorizablesConfig = new AuthorizablesConfig();
AuthorizableConfigBean authorizableConfigBeanSimple = new AuthorizableConfigBean();
String simpleAuthId = "testSimple";
String simpleAuthIdOld = "testSimpleOld";
authorizableConfigBeanSimple.setAuthorizableId(simpleAuthId);
authorizableConfigBeanSimple.setPrincipalName(simpleAuthId);
authorizableConfigBeanSimple.setMigrateFrom(simpleAuthIdOld);
authorizablesConfig.add(authorizableConfigBeanSimple);
AuthorizableConfigBean authorizableConfigBeanLdap = new AuthorizableConfigBean();
String ldapAuthId = "testLdap";
String ldapPrincipalName = "cn=testLdap,dc=name,dc=org";
String ldapAuthIdOld = "testLdapOld";
authorizableConfigBeanLdap.setAuthorizableId(ldapAuthId);
authorizableConfigBeanLdap.setPrincipalName(ldapPrincipalName);
authorizableConfigBeanLdap.setMigrateFrom(ldapAuthIdOld);
authorizablesConfig.add(authorizableConfigBeanLdap);
Set<String> principalsToBeMigrated = aceServiceImpl.collectPrincipalsToBeMigrated(authorizablesConfig);
// MethodAdderOnExistingObjectsAmplifier: added method on existing object
authorizableConfigBeanSimple.setSocialContent("QQmcN(DJ9-f?bfZ`LvH&");
// AssertionGenerator: add assertion
Assert.assertEquals("testSimple", simpleAuthId);
}
```

Open Source Contribution Study



RQ1: Deficiencies in amplified tests?

Selection Criteria

Manual Edits

RQ2: Feedback from maintainers on amplified tests?

Proposed Changes

Information Requested

Justification for Accept/Reject

Guidelines for Developers to Select and Edit Amplified Tests

Concern in Amplified Test	Explanation
Valid Coverage Improvement	Check that the targeted code is not tested by another test
Tests Relevant Code/Scenario in Project	Check that the new coverage provided by the test covers code that is relevant to test with your test suite
Only Necessary Code	Check that all code in the test is relevant for the test's execution or understandability
Checks Behavior of Newly Covered Code	Check that the assertion of the test actually validates the behavior of the additionally covered code
Test Scenario and Impact are Understandable	Check that you can / your colleagues could understand the test and what it is testing
Good Code Style, Adhering to Guidelines	Check that the code is well written and adheres to your guidelines
Appropriate Scope and Location	Check that the test is at an appropriate location and has the right granularity (move/merge/extend with other test otherwise)

Two kinds of edits / selections to perform on amplified tests:

(1) could be automated by customizing setup to project

(2) highly benefit from developer’s comprehension.

Should we focus on automating further or leave edits to developers and support them in understanding the tests?

Using GitHub Copilot for Test Generation in Python: An Empirical Study

Khalid El Haji, **Carolin Brandt**, Andy Zaidman

AST 2024

How usable are tests generated by generic code LLMs for developers?

Invoking Test Generations

Original
human-written test

```
def testProperties(self):
   '''Test all of the twitter.Status properties'''
   status = twitter.Status()
   status.id = 1
   self.assertEqual(1, status.id)
   created_at = calendar.timegm((2007, 1, 26, 23, 17, 14, -1, -1, -1))
   status.created_at = 'Fri Jan 26 23:17:14 +0000 2007'
   self.assertEqual('Fri Jan 26 23:17:14 +0000 2007', status.created_at)
   …
```

Stripped

```
def testProperties(self):
   '''Test all of the twitter.Status properties'’'
   …
```

Test generated
with Copilot

```
def testProperties(self):
   '''Test all of the twitter.Status properties'''
   status = self._GetSampleStatus()
   self.assertEqual('Fri Jan 26 23:17:14 +0000 2007', status.created_at)
   self.assertEqual(4391023, status.id)
   self.assertEqual(u'A légpárnás hajóm tele van angolnákkal.',
status.text)
   self.assertEqual(self._GetSampleUser(), status.user)
```

Project Selection + Study Execution

7 Open-source python projects

GitHub (Less popular ones) + GitLab

Generate batch of tests

Manually analyze label problems in generation

Repeat until theoretical saturation → 53 test pairs

Aspects of Usability

Syntactic Correctness

Runtime Correctness

Passing

Coverage

Variation 1: Invoking Generations Without a Test Suite

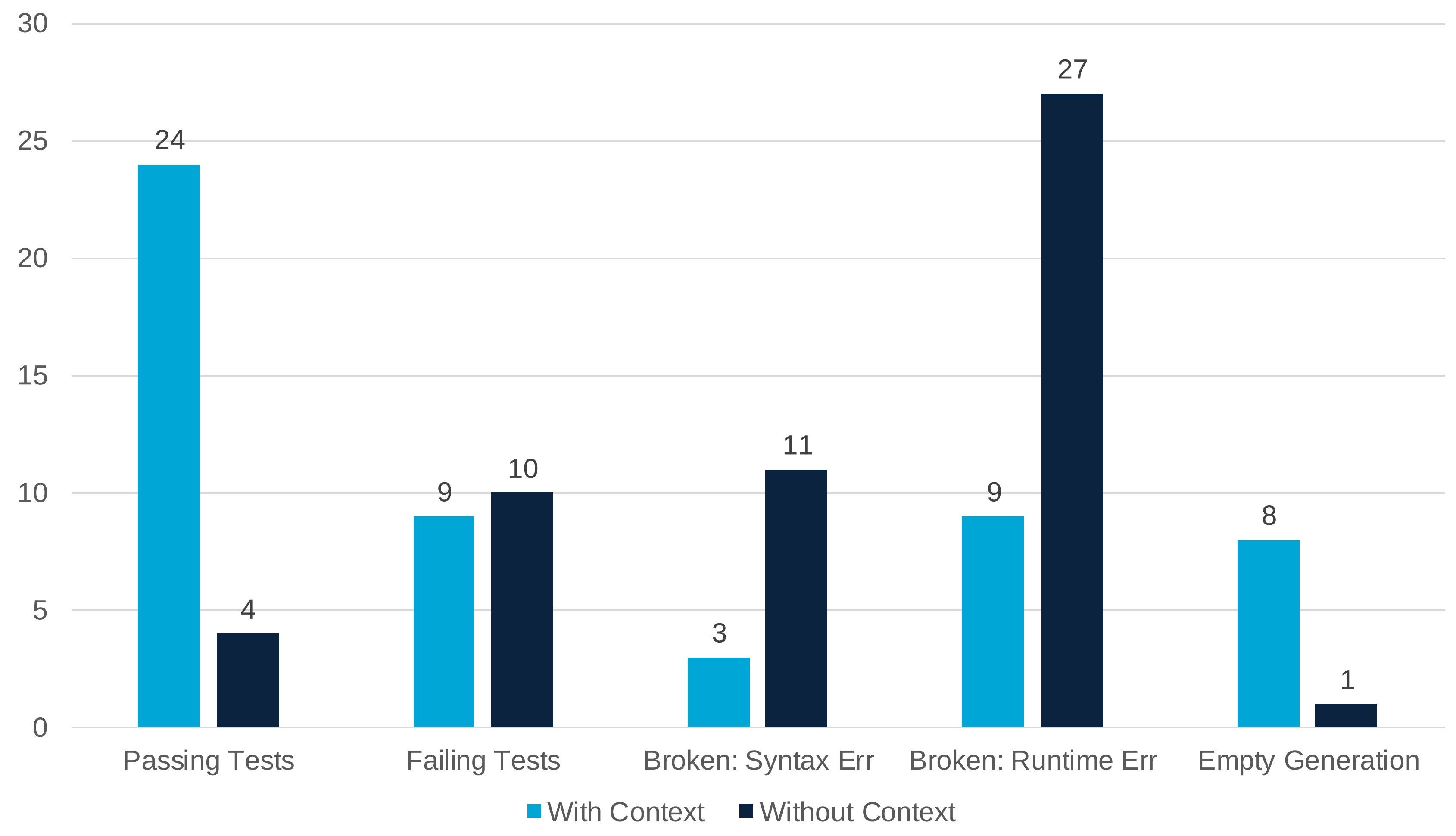
Original test code file (With-Context) [RQ1]

```
import pytest
from gcip import Cache, CacheKey, CachePolicy

def test_cache_policies():
   expected_members = ["PULL", "PULL_PUSH"]
   for member in CachePolicy.__members__:
       assert member in expected_members

def
test_default_cache_key_matches_ci_commit_ref_sl
ug():
   [INSERT]

def test_cache_key_with_custom_value():
   cache_key = CacheKey(key="mykey")
   expected_render = "mykey"
   assert expected_render == cache_key.render()
   assert cache_key.key == "mykey"
   assert cache_key.files is None
   assert cache_key.prefix is None
```

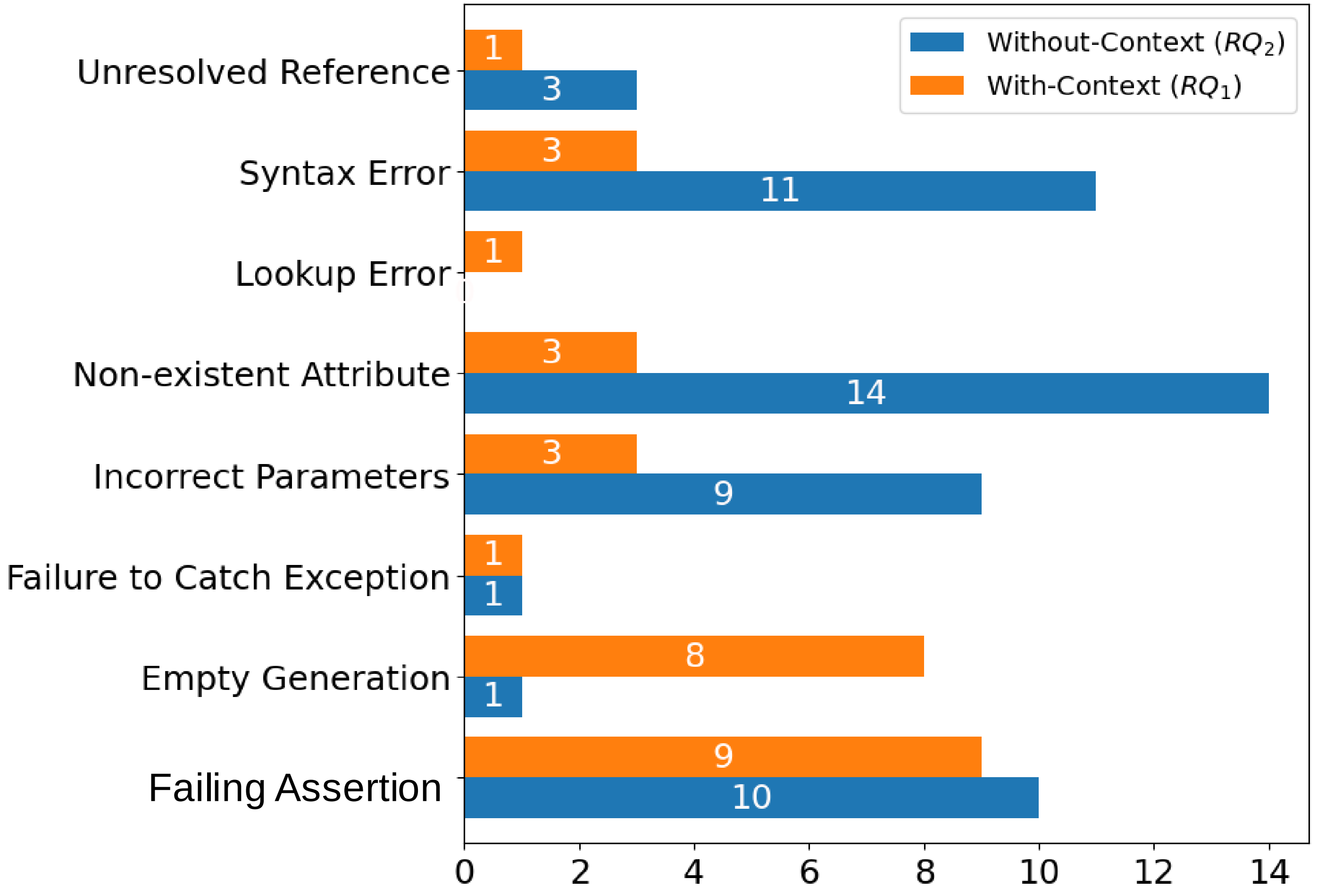
Stripped test code file (Without-Context) [RQ2]

```
import pytest
from gcip import Cache, CacheKey,
CachePolicy

def
test_default_cache_key_matches_ci_co
mmit_ref_slug():
```

Copilot Generations With + Without Testsuite Context



What were the problems?



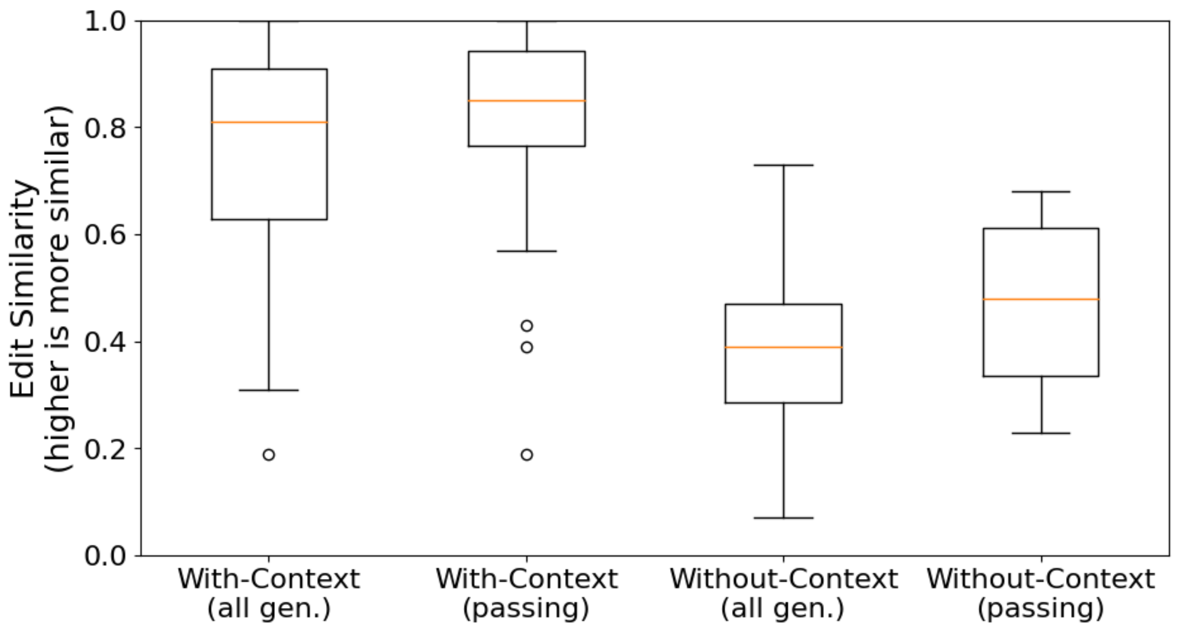
Observation: Mimicking Behavior

Test generated by Copilot

```
def test_option_optional():
    cli = Command("cli", params=[Option(["-c"], optional=True)])
    assert _get_words(cli, ["-c"], "") == []
    assert _get_words(cli, ["-c"], "-") == ["--help"]
```

Similar test in the same test file

```
def test_option_count():
    cli = Command("cli", params=[Option(["-c"], count=True)])
    assert _get_words(cli, ["-c"], "") == []
    assert _get_words(cli, ["-c"], "-") == ["--help"]
```



Variation 2: Different Test Method Comments

Minimal Method Comment (e.g, `"""Test the x function"""`)

Behavior-Driven Development Comment (e.g, `"""Given x when y then z"""`)

Usage Example Comment (e.g, `"""example: <code snippet> gives: <output>"""`)

Combined Comment

- For 23 tests where generation did not work

- Formulate comments based on original test

- Manually analyze problems again

Variation 2: Different Test Method Comments

With-Context	Minimal Method Comment	Behavior-Driven Development Comment	Usage Example Comment	Combined Comment
Passing	21.74%	26.09%	**34.78%**	26.09%
Failing	34.78%	30.43%	34.78%	34.78%
Broken	30.43%	30.43%	**17.39%**	26.09%
Empty	13.04%	13.04%	13.04%	13.04%

Without-Context	Minimal Method Comment	Behavior-Driven Development Comment	Usage Example Comment	Combined Comment
Passing	17.39%	13.04%	**21.74%**	**21.74%**
Failing	30.43%	26.09%	30.43%	47.83%
Broken	52.17%	60.87%	47.83%	**30.43%**
Empty	0.00%	0.00%	0.00%	0.00%

Takeaways

Generating tests **within an existing test suite**:

Poor usability, most generations will need to be edited

A code example in test method comments improves usability

Generations will likely mimic existing tests, can be useful for writing repetitive tests

Generating tests **without an existing test suite**:

Extremely poor usability, almost all generations will need to be edited

Instructive natural language combined with a code example in test method comments improves usability

Using GitHub Copilot for Test Generation in Python: An Empirical Study

Khalid El Haji, Carolin Brandt, Andy Zaidman

Recap

Software testing is …

Slow

Painful

Boring

Necessary

Automation to the rescue !

TU Delft

We built many ways to generate tests…

Generate Test Data

Fuzzing

Symbolic Execution

Model Checking

Generate Test Code

Random Testing

Search-based / Genetic Algorithms

AI / ML-based

Test Carving

Hybrid

Test Amplification

TU Delft

Test Generation Tools

Quite dominated by research tools….

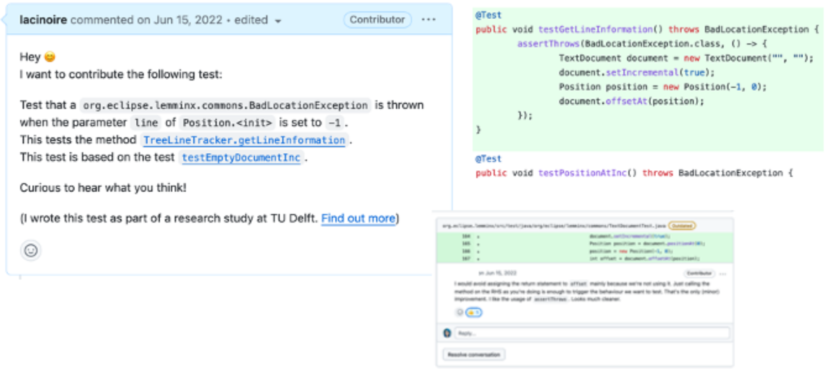
- **TestSpark IntelliJ Plugin** (https://plugins.jetbrains.com/plugin/21024-testspark, https://github.com/JetBrains-Research/TestSpark): Java and Kotlin, LLM (openAI, internal JetBrains AI) & Search-based (local, EvoSuite)
- **GitHub Test Pilot** (https://githubnext.com/projects/testpilot/): JavaScript / TypeScript, LLM (GitHub Copilot)
- **TestCube IntelliJ Plugin** (https://plugins.jetbrains.com/plugin/14678-test-cube): Java, Test Amplification (local, DSpot)

Lots of detailed / CLI tools:

EvoSuite (search-based, Java), Pyguin (search-based, python), DSpot (test amplification, Java),
Plenty Fuzzing tools (https://github.com/secfigo/Awesome-Fuzzing)

TU Delft

Open Source Contribution Study



lacinoire commented on Jun 15, 2022 • edited — Contributor

Hey 😄
I want to contribute the following test:

Test that a `org.eclipse.lemminx.commons.BadLocationException` is thrown when the parameter `line` of `Position.<init>` is set to `-1`.
This tests the method `TreeLineTracker.getLineInformation`.
This test is based on the test `testEmptyDocumentInc`.

Curious to hear what you think!

(I wrote this test as part of a research study at TU Delft. Find out more)

```java
@Test
public void testGetLineInformation() throws BadLocationException {
        assertThrows(BadLocationException.class, () -> {
                TextDocument document = new TextDocument("", "");
                document.setIncremental(true);
                Position position = new Position(-1, 0);
                document.offsetAt(position);
        });
}

@Test
public void testPositionAtInc() throws BadLocationException {
```

How usable are tests generated by generic code LLMs for developers?

Should we focus on automating further or leave edits to developers and support them in understanding the tests?

?

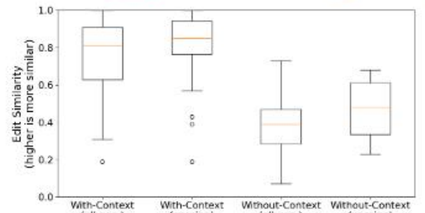
Observation: Mimicking Behavior

Test generated by Copilot

```python
def test_option_optional():
    cli = Command("cli", params=[Option(["-c"], optional=True)])
    assert _get_words(cli, ["-c"], "") == []
    assert _get_words(cli, ["-c"], "-") == ["--help"]
```

Similar test in the same test file

```python
def test_option_count():
    cli = Command("cli", params=[Option(["-c"], count=True)])
    assert _get_words(cli, ["-c"], "") == []
    assert _get_words(cli, ["-c"], "-") == ["--help"]
```



Sources

- Motivation and first slides on types of test generation inspired by Annibale Panichella's slides for hi VST 22 talk: https://apanichella.github.io/talk/do-tests-generated-by-ai-help-developers-open-challenges-applications-and-opportunities/