Priorisierung von Quelltextduplikaten in Testcode durch die Kombination von Clone-Detection und testspezischer Coverage

Stefan Knilling	CQSE GmbH / Technische Universität München
Jakob Rott	CQSE GmbH
Roman Haas	CQSE GmbH

Agenda

1. **Ausgangssituation**
2. Ansatz
3. Fallstudie
4. Fazit

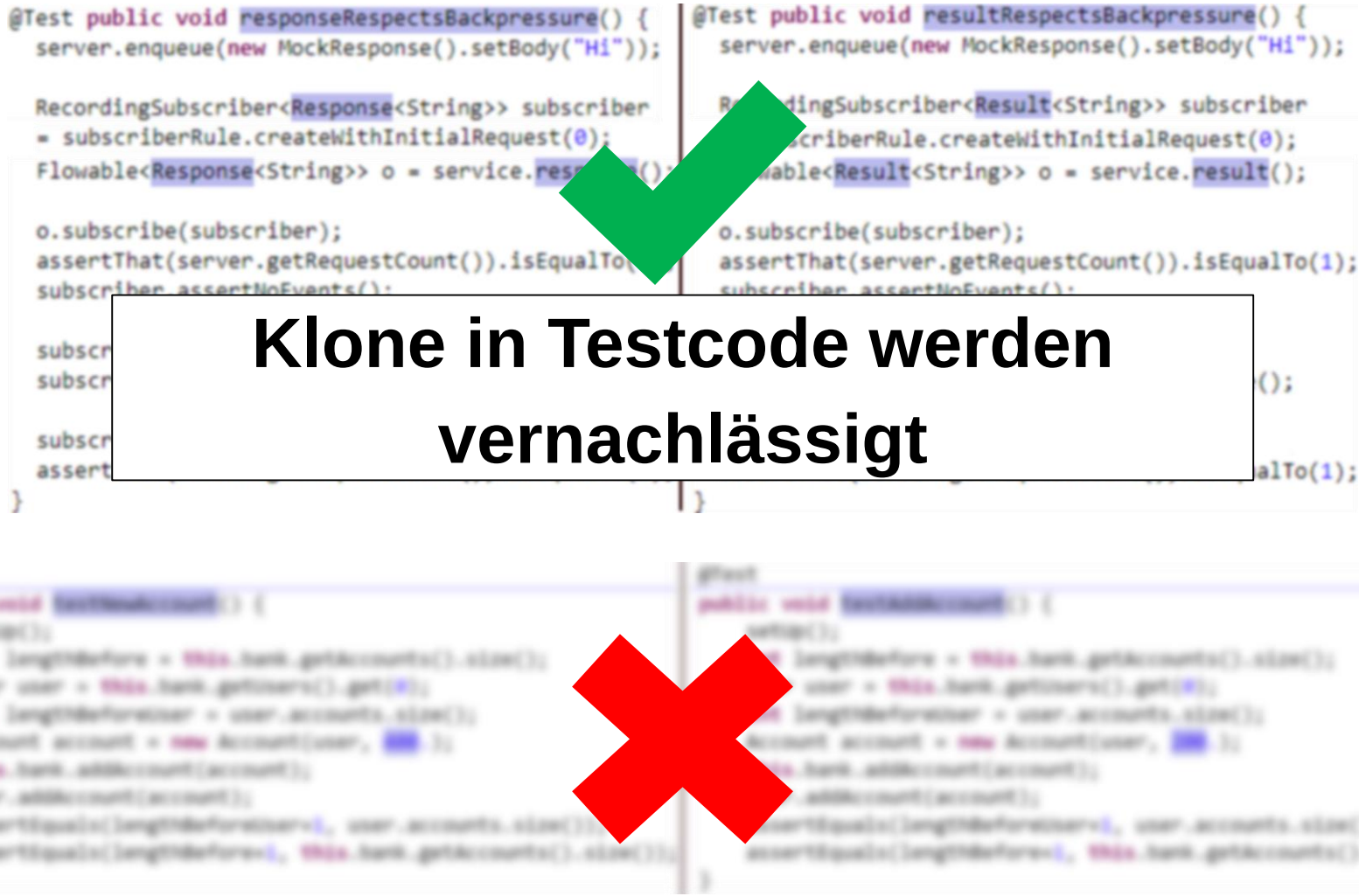
1.1 Code Klone

```java
@Test public void responseRespectsBackpressure() {
  server.enqueue(new MockResponse().setBody("Hi"));

  RecordingSubscriber<Response<String>> subscriber
  = subscriberRule.createWithInitialRequest(0);
  Flowable<Response<String>> o = service.response();

  o.subscribe(subscriber);
  assertThat(server.getRequestCount()).isEqualTo(1);
  subscriber.assertNoEvents();

  subscriber.request(1);
  subscriber.assertAnyValue().assertComplete();

  subscriber.request(Long.MAX_VALUE);
  assertThat(server.getRequestCount()).isEqualTo(1);
}
```

```java
@Test public void resultRespectsBackpressure() {
  server.enqueue(new MockResponse().setBody("Hi"));

  RecordingSubscriber<Result<String>> subscriber
  = subscriberRule.createWithInitialRequest(0);
  Flowable<Result<String>> o = service.result();

  o.subscribe(subscriber);
  assertThat(server.getRequestCount()).isEqualTo(1);
  subscriber.assertNoEvents();

  subscriber.request(1);
  subscriber.assertAnyValue().assertComplete();

  subscriber.request(Long.MAX_VALUE);
  assertThat(server.getRequestCount()).isEqualTo(1);
}
```

- Höherer Wartungsaufwand
- Fehler aufgrund inkonsistenter Änderungen

1.2 Code Klone in Testcode



1. 3 Testspezifische Code Coverage

zxing / com / google / zxing / client / result / TelParsedResultTestCase / **testTel**

PASSED

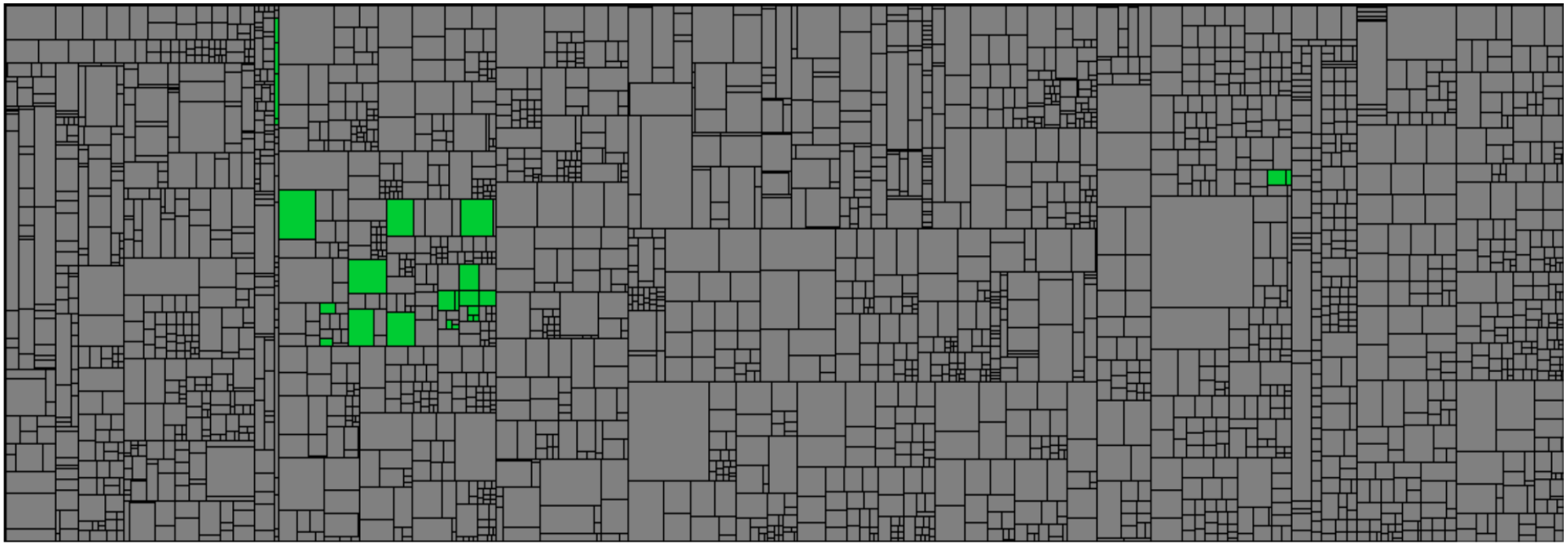
master@Nov 14 2019 16:50

Partition **testwise-coverage**

0.025s

Executed Methods

Show only executed methods.



1.4 Code Coverage geklonter Testfälle

```java
@Test public void responseRespectsBackpressure() {
  server.enqueue(new MockResponse().setBody("Hi"));

  RecordingSubscriber<Response<String>> subscriber
  = subscriberRule.createWithInitialRequest(0);
  Flowable<Response<String>> o = service.response();

  o.subscribe(subscriber);
  assertThat(server.getRequestCount()).isEqualTo(1);
  subscriber.assertNoEvents();

  subscriber.request(1);
  subscriber.assertAnyValue().assertComplete();

  subscriber.request(Long.MAX_VALUE);
  assertThat(server.getRequestCount()).isEqualTo(1);
}
```

1

Geklont?

```java
@Test public void resultRespectsBackpressure() {
  server.enqueue(new MockResponse().setBody("Hi"));

  RecordingSubscriber<Result<String>> subscriber
  = subscriberRule.createWithInitialRequest(0);
  Flowable<Result<String>> o = service.result();

  o.subscribe(subscriber);
  assertThat(server.getRequestCount()).isEqualTo(1);
  subscriber.assertNoEvents();

  subscriber.request(1);
  subscriber.assertAnyValue().assertComplete();

  subscriber.request(Long.MAX_VALUE);
  assertThat(server.getRequestCount()).isEqualTo(1);
}
```

2

Ähnliche Coverage?

Agenda

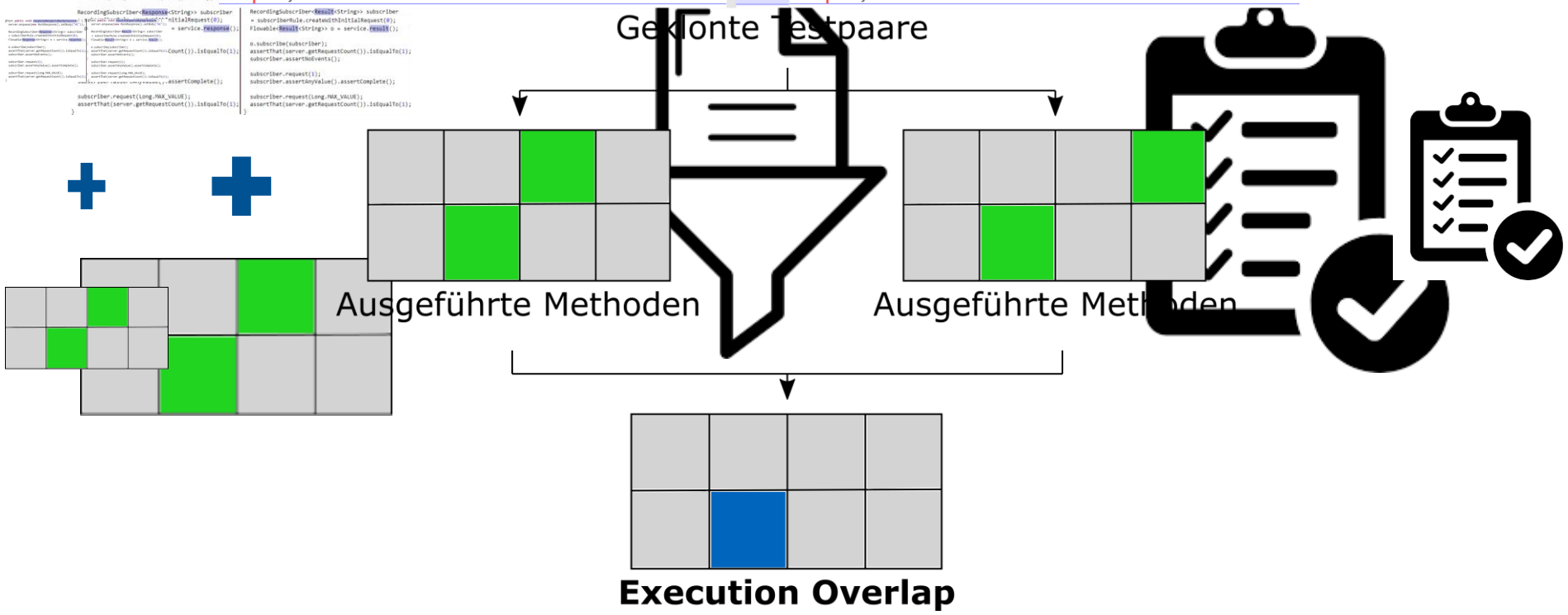
1. Ausgangssituation
2. **Ansatz**
3. Fallstudie
4. Fazit

2. Ansatz

I In ut O

```
42      @Test
43      public void testNewAccount() {
44          setUp();
45          int lengthBefore = this.bank.getAccounts().size();
46          User user = this.bank.getUsers().get(0);
47          int lengthBeforeUser = user.accounts.size();
48          Account account = new Account(user, 400.);
49          this.bank.addAccount(account);
50          user.addAccount(account);
51          assertEquals(lengthBeforeUser+1, user.accounts.size());
52          assertEquals(lengthBefore+1, this.bank.getAccounts().size());
53      }
```

```
28      @Test
29      public void testAddAccount() {
30          setUp();
31          int lengthBefore = this.bank.getAccounts().size();
32          User user = this.bank.getUsers().get(0);
33          int lengthBeforeUser = user.accounts.size();
34          Account account = new Account(user, 200.);
35          this.bank.addAccount(account);
36          user.addAccount(account);
37          assertEquals(lengthBeforeUser+1, user.accounts.size());
38          assertEquals(lengthBefore+1, this.bank.getAccounts().size());
39      }
```



2.1 Relevanz von Code Klonen

Annahmen:

Hoher Execution Overlap ⟺ Hohe Relevanz des Klons

Niedriger Execution Overlap ⟺ Niedrige Relevanz des Klons

Relevanz eines Code Klons:

- Ein Entwickler möchte den Klon refaktorieren.
  UND
- Ein Entwickler möchte von einem automatischen Verfahren einen Hinweis auf den Klon erhalten.

Agenda

1. Ausgangssituation
2. Ansatz
3. **Fallstudie**
4. Fazit

3. Fallstudie

RQ1

Ausmaß von Klonen in Test- und Produktivcode

- Clone Coverage

RQ2

Korrelation von Execution Overlap und Relevanz

- Entwicklerbefragung
- Vergleich des Ergebnisses der Befragung mit dem Execution Overlap

3.1 Studienobjekte und Studiendesign

9 Open Source Java Projekte

- 20 – 200 kLOC
- Variierende Reputation und Anzahl an Mitwirkenden

40 Paare geklonter Tests

- Zufällig ausgewählt
- Schätzung der Relevanz in Entwicklerbefragung

3.2 Technisches Setup

JaCoCo

- Testspezifische Line Coverage

Teamscale

- Method Coverage
- Clone Detection
- Weitere Projektkennzahlen

Weitere Tools

- Transformieren der Daten aus JaCoCo
- Kombinieren der Code Coverage und der Klon Informationen

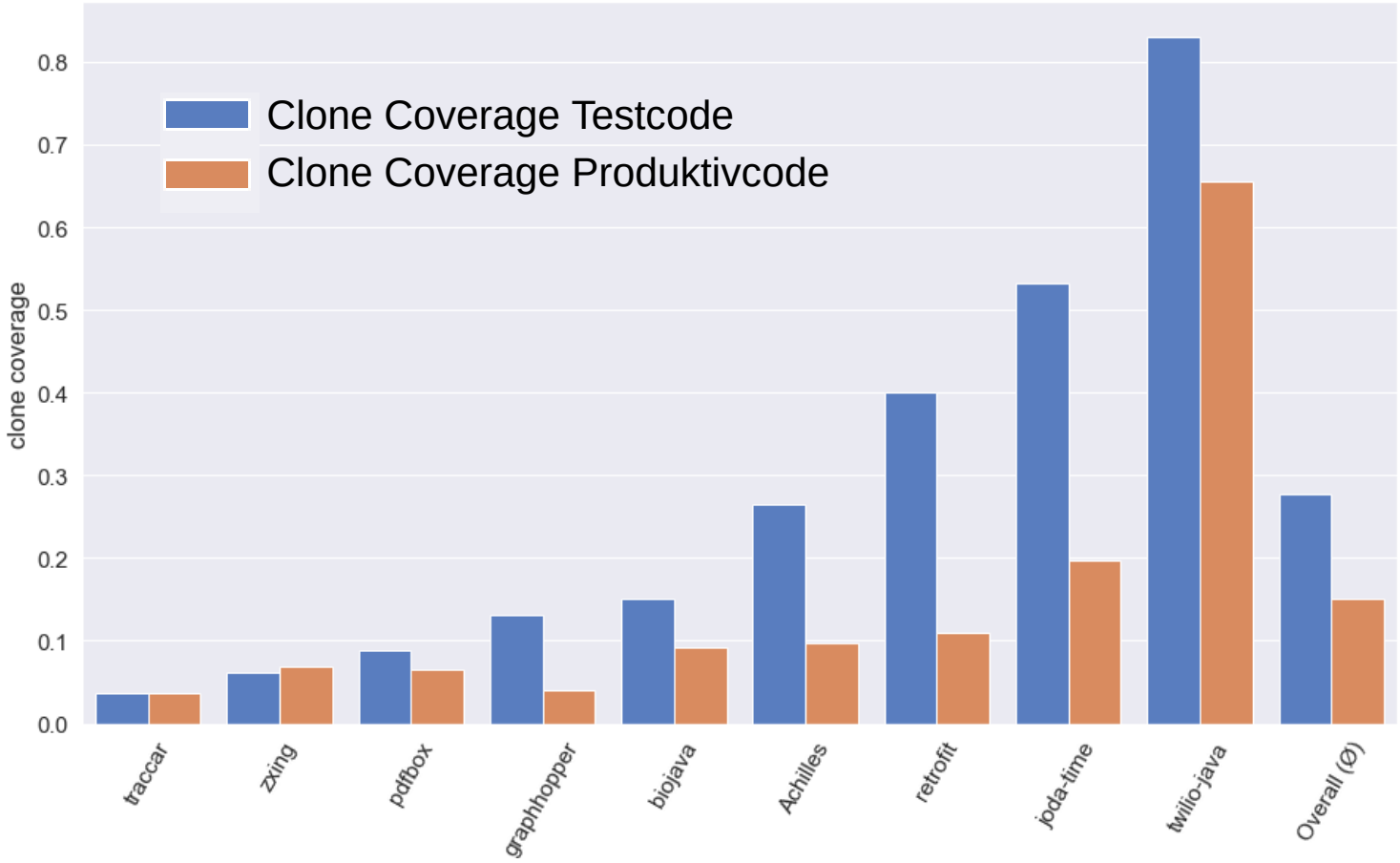
3.3 Klone in Test- und Produktivcode

RQ1

Ø Clone Coverage Produktivcode: 15,2%

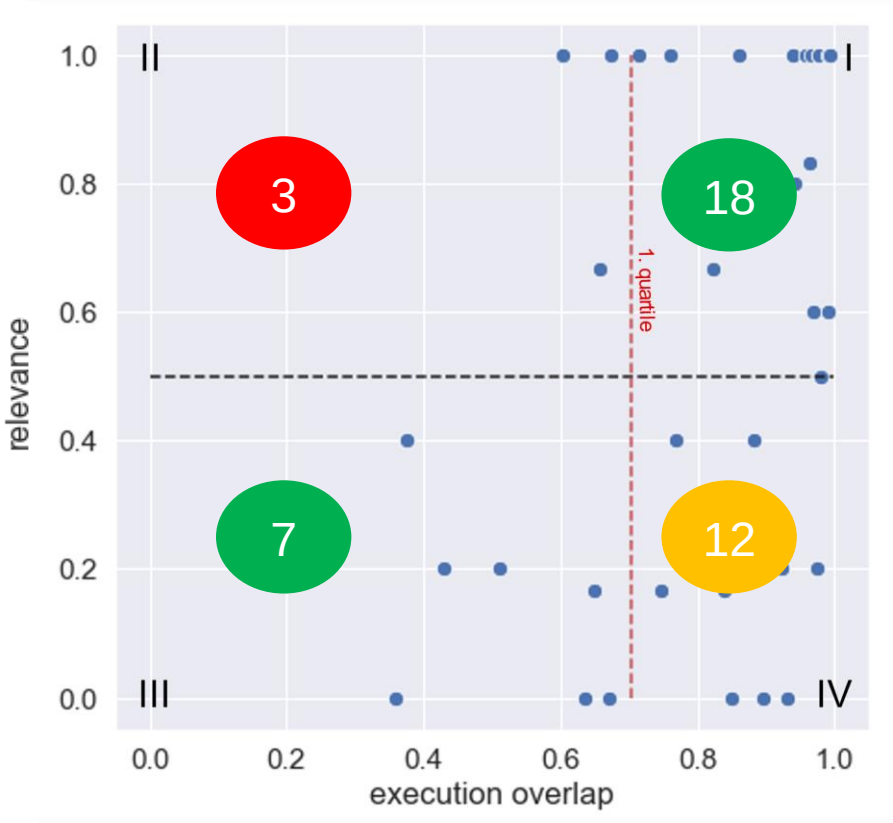
Ø Clone Coverage Testcode: 27,7%

+ 12.5 pp



3.4 Execution Overlap und Relevanz (I)

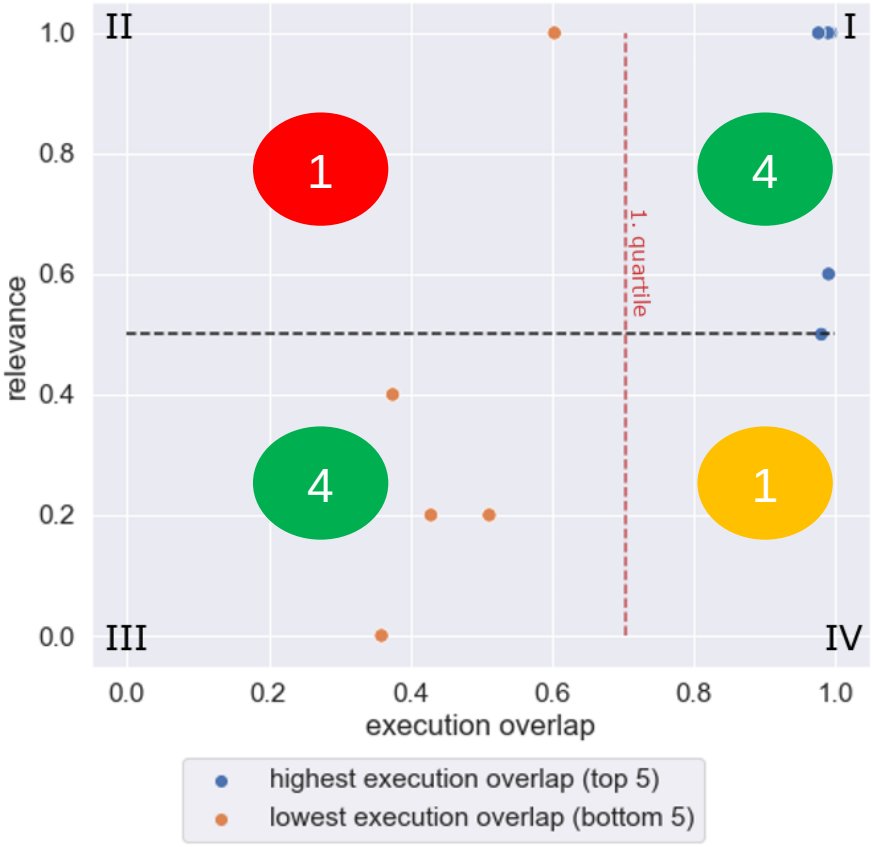
RQ2



Rangkorrelation (Spearman): 0,364

3.4 Execution Overlap und Relevanz (II)

RQ2



Rangkorrelation (Spearman): 0,747

Agenda

1. Ausgangssituation
2. Ansatz
3. Fallstudie
4. **Fazit**

4. Fazit

RQ1

Klone in Testcode sind üblich

RQ2

Execution Overlap kann ein Indikator für die Relevanz sein

- Nicht stark genug um Klone zu filtern
- Alternative: Priorisierung von Klonen in Abhängigkeit des Execution Overlap

Vielen Dank!