



Developer Survey:

Automated test + mock generation with **RICK**

Deepika Tiwari, Martin Monperrus, Benoit Baudry

Agenda

- Introduction
- Tests generated by **RICK**
- Discussion

Introduction: Mocking

```
public class ReservationCentre {  
  
    . . .  
  
    public ReservationStatus reserve(int qty, PaymentService paymentService) {  
  
        if (paymentService.checkActiveConnections() > 0) {  
            boolean isPaymentSuccessful = paymentService.processPayment(qty);  
            . . .  
            return ReservationStatus.SUCCESS;  
        }  
  
        return ReservationStatus.FAIL;  
    }  
  
}
```

```
public class ReservationCentre {
```

```
    . . .
```

```
    public ReservationStatus reserve(int qty, PaymentService paymentService) {  
        if (paymentService.checkActiveConnections() > 0) {  
            boolean isPaymentSuccessful = paymentService.processPayment(qty);  
            . . .  
            return ReservationStatus.SUCCESS;  
        }  
  
        return ReservationStatus.FAIL;  
    }
```

```
}
```

```
public class ReservationCentre {
```

```
. . .
```

```
public ReservationStatus reserve(int qty, PaymentService paymentService) {  
    if (paymentService.checkActiveConnections() > 0) {  
        boolean isPaymentSuccessful = paymentService.processPayment(qty);  
        . . .  
        return ReservationStatus.SUCCESS;  
    }  
  
    return ReservationStatus.FAIL;  
}
```

```
}
```

```
public class ReservationCentre {
```

```
. . .
```

```
public ReservationStatus reserve(int qty, PaymentService paymentService) {  
    if (paymentService.checkActiveConnections() > 0) {  
        boolean isPaymentSuccessful = paymentService.processPayment(qty);  
        . . .  
        return ReservationStatus.SUCCESS;  
    }  
  
    return ReservationStatus.FAIL;  
}
```

```
}
```

MUT

mockable

```
@Test
public void testThatReservationIsMade() {

    // Arrange
    ReservationCentre centre = new ReservationCentre();

    PaymentService mockPaymentService = mock(PaymentService.class);
    when(mockPaymentService.checkActiveConnections()).thenReturn(3);
    when(mockPaymentService.processPayment(42.42)).thenReturn(true);

    // Act
    ReservationStatus status = centre.reserve(6, mockPaymentService);

    // Assert
    assertEquals(ReservationStatus.SUCCESS, status);
    verify(mockPaymentService, times(1)).checkActiveConnections();
    verify(mockPaymentService, times(1)).processPayment(anyDouble());
}
```



```
@Test
public void testThatReservationIsMade() {

    // Arrange
    ReservationCentre centre = new ReservationCentre();

    PaymentService mockPaymentService = mock(PaymentService.class);
    when(mockPaymentService.checkActiveConnections()).thenReturn(3);
    when(mockPaymentService.processPayment(42.42)).thenReturn(true);

    // Act
    ReservationStatus status = centre.reserve(6, mockPaymentService);

    // Assert
    assertEquals(ReservationStatus.SUCCESS, status);
    verify(mockPaymentService, times(1)).checkActiveConnections();
    verify(mockPaymentService, times(1)).processPayment(anyDouble());
}
```

MOCK

```
@Test
public void testThatReservationIsMade() {

    // Arrange
    ReservationCentre centre = new ReservationCentre();

    PaymentService mockPaymentService = mock(PaymentService.class);
    when(mockPaymentService.checkActiveConnections()).thenReturn(3);
    when(mockPaymentService.processPayment(42.42)).thenReturn(true);

    // Act
    ReservationStatus status = centre.reserve(6, mockPaymentService);

    // Assert
    assertEquals(ReservationStatus.SUCCESS, status);
    verify(mockPaymentService, times(1)).checkActiveConnections();
    verify(mockPaymentService, times(1)).processPayment(anyDouble());
}
```

STUB

```
@Test
public void testThatReservationIsMade() {

    // Arrange
    ReservationCentre centre = new ReservationCentre();

    PaymentService mockPaymentService = mock(PaymentService.class);
    when(mockPaymentService.checkActiveConnections()).thenReturn(3);
    when(mockPaymentService.processPayment(42.42)).thenReturn(true);

    // Act
    ReservationStatus status = centre.reserve(6, mockPaymentService);

    // Assert
    assertEquals(ReservationStatus.SUCCESS, status);
    verify(mockPaymentService, times(1)).checkActiveConnections();
    verify(mockPaymentService, times(1)).processPayment(anyDouble());
}
```

VERIFY

```
@Test
public void testThatReservationIsMade() {

    // Arrange
    ReservationCentre centre = new ReservationCentre();

    PaymentService mockPaymentService = mock(PaymentService.class);
    when(mockPaymentService.checkActiveConnections()).thenReturn(3);
    when(mockPaymentService.processPayment(42.42)).thenReturn(true);

    // Act
    ReservationStatus status = centre.reserve(6, mockPaymentService);

    // Assert
    assertEquals(ReservationStatus.SUCCESS, status);

    verify(mockPaymentService, times(1)).checkActiveConnections();
    verify(mockPaymentService, times(1)).processPayment(anyDouble());

}
```

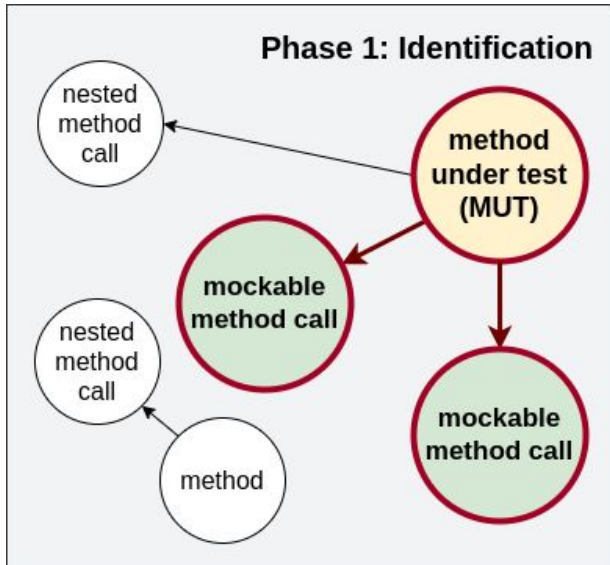
MOCK

STUB

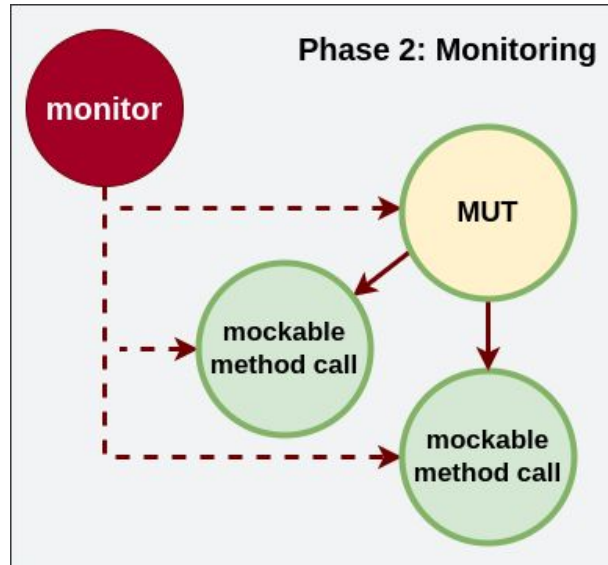
VERIFY

Introduction: RICK

Key phases



Key phases



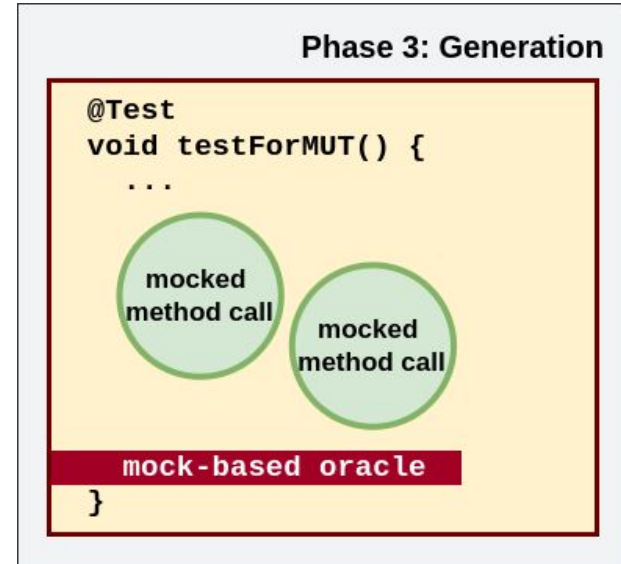
Key phases

Phase 3: Generation

```
@Test  
void testFormUT() {  
    ...  
    mocked  
    method call  
    mocked  
    method call  
    mock-based oracle  
}
```

The diagram illustrates the 'Generation' phase of a testing process. It features a light yellow rectangular area with a dark red border. Inside this area, at the top, is the code snippet: `@Test`, `void testFormUT() {`, `...`, and `}`. Below the code, there are two light green circles, each containing the text 'mocked method call'. At the bottom of the yellow area, there is a dark red horizontal bar with the text 'mock-based oracle' in white. The entire yellow area is set against a light gray background.

Key phases



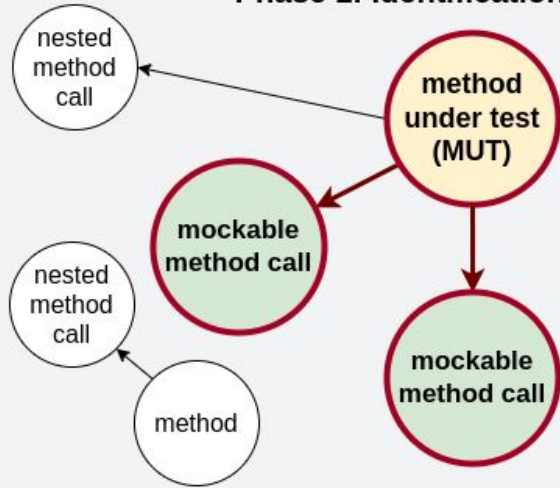
OO: Output oracle

PO: Parameter oracle

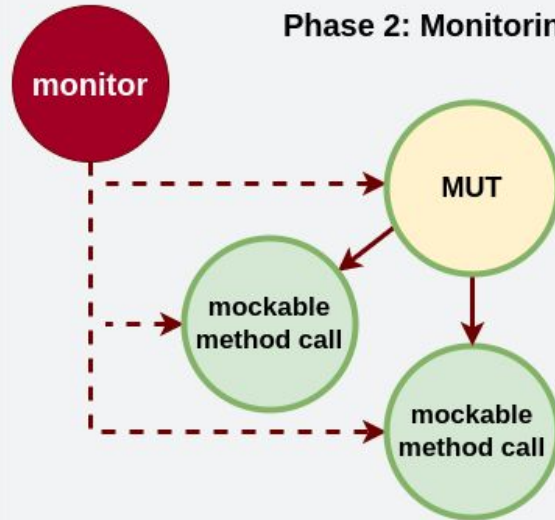
CO: Call oracle

Key phases

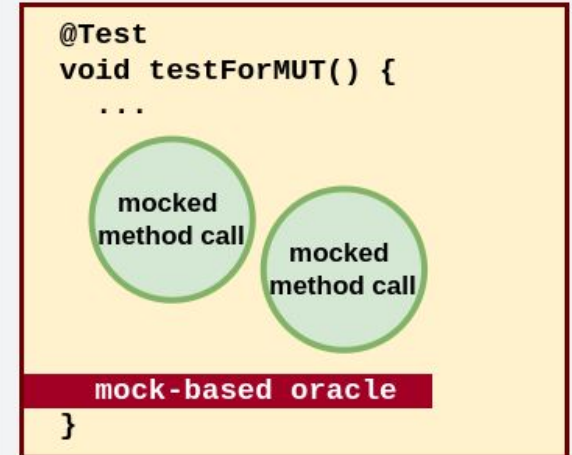
Phase 1: Identification



Phase 2: Monitoring



Phase 3: Generation



Introduction: **Graphhopper** maps

- <https://www.graphhopper.com/>
- Routing: Point A to B (to C...)
- On bike, foot, car, (mountain bike, ...)
- Elevation, layers, ...
- <https://graphhopper.com/maps/>

Experiment

RICK + Graphhopper maps

1  Kungsgatan, Uppsala

2  Virebergsvägen, Stockholm






3  Gröndalsvägen, Stockholm

4  Kungliga Tekniska Högskolan, Stock

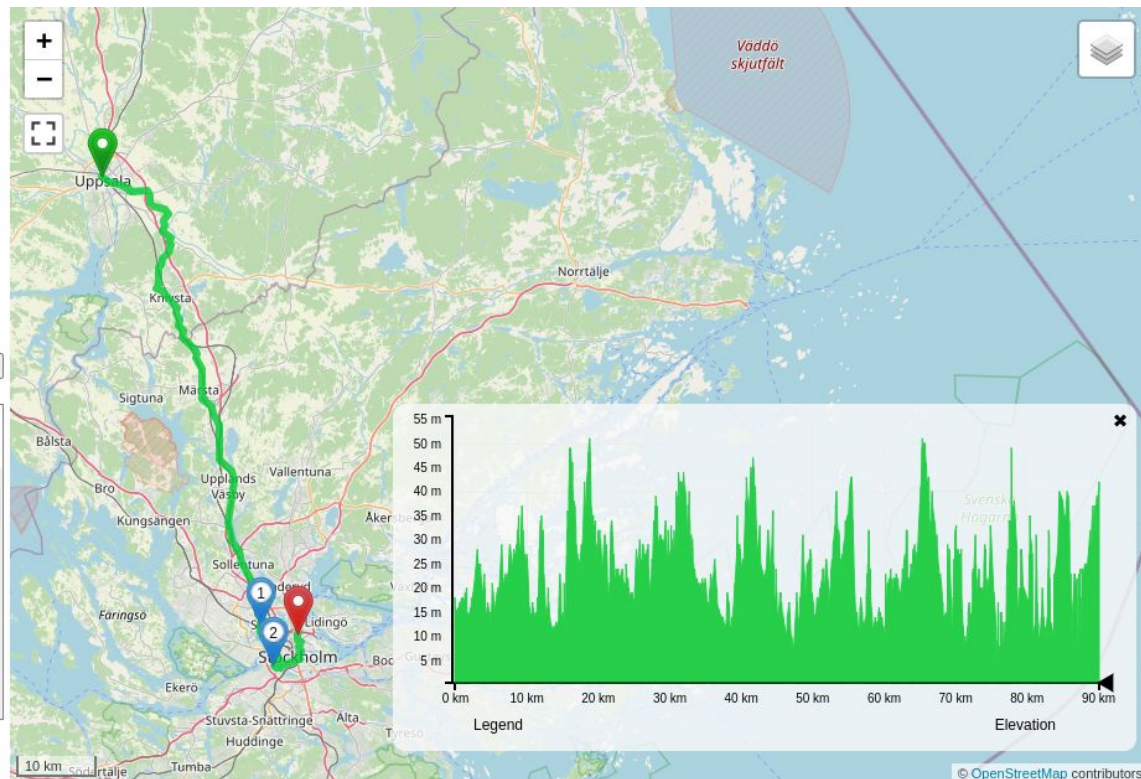


custom  gpx

Search

90.39km will take 5h 2min	km mi
↗1406m ↘1379m	
 Continue onto Kungsgatan	10m 0min
 Turn right onto Skolgatan	524m 2min
 Turn left onto Sankt Persgatan	115m 0min
 Turn right onto Storgatan	339m 1min
 Turn left onto ...	640m

[Donate](#) [Imprint](#) [Terms](#) [Privacy](#)



Generated tests

Clone

```
git clone git@github.com:Deee92/graphhopper.git --branch 5.3-rick
```

Browse on 

<https://github.com/Deee92/graphhopper/tree/5.3-rick>

MUT #1

MUT:

`com.graphhopper.util.Instruction.getTurnDescription(Translation)`



Mockable:

`com.graphhopper.util.Translation.tr(String, Object...)`



Generated tests:

`com.graphhopper.util.TestInstructionRickGen`



MUT #2

MUT:

```
com.graphhopper.storage.StorableProperties.loadExisting()
```



Mockable:

```
com.graphhopper.storage.DataAccess.loadExisting()
```

```
com.graphhopper.storage.DataAccess.getCapacity()
```

```
com.graphhopper.storage.DataAccess.getBytes(long, byte[], int)
```



Generated tests:

```
com.graphhopper.storage.TestStorablePropertiesRickGen
```



Discussion

Mocking effectiveness: P1

Statement	Response
I would mock the same objects myself	yes
I would write similar oracles myself	Yes, but all oracles in one test
The oracles represent realistic values which would be useful to developers	Represent actual production behavior but more obvious to developers (e.g., “continue onto Kungsgatan” in MUT1)
The use of mocks allows for more focus on the MUT	yes

Mocking effectiveness: P2

Statement	Response
I would mock the same objects myself	For the muts, would probably mock; Won't mock package-internal or legacy, depends on domain knowledge (how big class is)
I would write similar oracles myself	Probably, but depends on the data, not sure about CO (if performance critical)
The oracles represent realistic values which would be useful to developers	Realistic data, saves time, deciding inputs, corner cases - combinations of parameters, Would typically use my own data
The use of mocks allows for more focus on the MUT	yes

Mocking effectiveness: P3

Statement	Response
I would mock the same objects myself	Prefer to define test doubles manually, not use mocking frameworks, if possible real collaborator, depends on complexity of the collaborator (IO, networking), more control over tests
I would write similar oracles myself	OO is strongest, PO, CO are verifications on different levels, PO stronger than CO
The oracles represent realistic values which would be useful to developers	Make sense for examples (MUTs), I try to avoid primitive types, use complex types
The use of mocks allows for more focus on the MUT	Rigid snapshot is useful, but would complicate refactoring, Not for me, would make test doubles myself

Mocking effectiveness: P4

Statement	Response
I would mock the same objects myself	MUTs can be trivial, but good examples to understand RICK
I would write similar oracles myself	The different oracles make sense, hybrid = manual + RICK, Maybe put in the same test
The oracles represent realistic values which would be useful to developers	Yes, realistic
The use of mocks allows for more focus on the MUT	Mockable without reflection: code is good

Mocking effectiveness: P5

Statement	Response
I would mock the same objects myself	Absolutely, focus on external objects as parameters, attributes of class - absolutely necessary, identification phase very useful
I would write similar oracles myself	Yes, caveat: some statements in PO tests for methods that don't take parameters are redundant, can complicate troubleshooting if tests fail
The oracles represent realistic values which would be useful to developers	Data collected from production, RICK shines most - does not get more accurate than production data, tricky exercise and time consuming to come up with input values and internal states, RICK completely abstracts this away from developer
The use of mocks allows for more focus on the MUT	Understanding MUT - laser focused unit test to avoid redundancy - extremely expensive and time consuming - value in writing tests manually. Ideally, we strive for TDD - big concern, application already functional production-ready - RICK would be redundant because developers have already written tests In real-world it is rare, focus on testability Fits a real Industry problem exactly, get test coverage Thinking about testability during design is an important problem - spoiling developers, make testing easier with automated framework

Structure: P1

Statement	Response
The tests have a clear structure	AAA is very clear, display info
I would follow a similar structure	Maybe, but Input(AA)-oracle(A) path can also be followed
Suggestions for improvement: name of the test (snake case), display info can also specify the return type, flag for kind of oracle or merge them	

Structure: P2

Statement	Response
The tests have a clear structure	Yes, comprehensible, display name is rare and useful
I would follow a similar structure	Yes, Setup teardown cleanup, Null values and corner cases, crashes
Suggestions for improvement: not at the moment, before after	

Structure: P3

Statement	Response
The tests have a clear structure	Yes, very clear structure
I would follow a similar structure	Yes, use AAA
Suggestions for improvement: extract similar statements in a setup phase / delegated to other method	

Structure: P4

Statement	Response
The tests have a clear structure	Yes, Maybe remove the comment
I would follow a similar structure	Setup etc important, good for generated tests Maybe not constrained for all manually written tests
Suggestions for improvement: displayname could be long for more mocked methods	

Structure: P5

Statement	Response
The tests have a clear structure	Yes very clear structure, additional comments super easy to visualize
I would follow a similar structure	Absolutely yes, spot on
Suggestions for improvement: name of the methods and comments logic business logic - helpful when tests fail, what to expect and why, automatically generated spot but do not understand, tricky to give business logic + context, so more important to have laser-focused oracles, non-redundant, exactly what went wrong when tests fail	

Understandability: P1

Statement	Response
The tests are generally intuitive / understandable	Yes, easy to read because of structure
The intention of each test is clear	Yes, PO and CO are different, contribute differently to verification
Suggestions for improvement: N/A	

Understandability: P2

Statement	Response
The tests are generally intuitive / understandable	Yes, very, Data variations, more data for method with branches
The intention of each test is clear	Yes from display name, pattern AAA
Suggestions for improvement: not at the moment	

Understandability: P3

Statement	Response
The tests are generally intuitive / understandable	Yes, due to AAA
The intention of each test is clear	Extract shared code into a method and call that. Assertion phase: CO vs PO is not clear (Java / Python), types
Suggestions for improvement: Comments not necessary - line break suffices for AAA	

Understandability: P4

Statement	Response
The tests are generally intuitive / understandable	yes
The intention of each test is clear	yes
Suggestions for improvement: this is fine / good	

Understandability: P5

Statement	Response
The tests are generally intuitive / understandable	Generally clear but sometimes redundant, naming and comments are agnostic to business logic, more time to troubleshoot, can be improved
The intention of each test is clear	Yes
Suggestions for improvement: name + comments add context, redundancy of oracles, PO and CO are valuable, and it makes sense to keep them separate, if manually writing tests would probably combine them, but for automated, each oracle is distinct, allows for laser-focus and clear understanding	

Profile: P1

Years of experience with software development	6 years
Position in team / organisation	Researcher, software that consumes less energy, exploiting software tests as production traffic
Size of team / organisation	4

Mocking practices: P1

- How often do you write tests?
 - **Often, everyday, testing is life**
 - Sometimes
 - Rarely
- How often do you create / use mocks?
 - Often
 - **Sometimes**
 - Rarely
- How familiar are you with Java + JUnit + Mockito?
 - **Proficient**
 - **Comfortable** - Mockito
 - Novice

Profile: P2

Years of experience with software development	30-40 (1982 first computer)
Position in team / organisation	Test environment architect
Size of team / organisation	22 in department, 8k in bank

Mocking practices: P2

- How often do you write tests?
 - **Often**
 - Sometimes
 - Rarely
- How often do you create / use mocks?
 - Often
 - **Sometimes**
 - Rarely
- How familiar are you with Java + JUnit + Mockito?
 - **Proficient**
 - Comfortable
 - Novice

Profile: P3

Years of experience with software development	7 years
Position in team / organisation	Developer, technical coach
Size of team / organisation	Team: 6, org: 220k

Mocking practices: P3

- How often do you write tests?
 - **Often - TDD**
 - Sometimes
 - Rarely
- How often do you create / use mocks?
 - Often: test doubles, fake collaborators
 - **Sometimes**
 - Rarely: mocks
- How familiar are you with Java + JUnit + Mockito?
 - Proficient
 - Comfortable
 - Novice
 - **N/A: I work with pytest (mocks included)**

Profile: P4

Years of experience with software development	~10 (5 with Graphhopper)
Position in team / organisation	Developer
Size of team / organisation	3 (6)

Mocking practices: P4

- How often do you write tests?
 - **Often, all the time**
 - Sometimes
 - Rarely
- How often do you create / use mocks?
 - Often
 - **Sometimes, define mock classes / dummy implementation**
 - **Rarely, Graphhopper does not use mockito**
- How familiar are you with Java + JUnit + Mockito?
 - **Proficient**
 - Comfortable
 - Novice

Profile: P5

Years of experience with software development	14 years
Position in team / organisation	Director of Engineering, Group Technical Director
Size of team / organisation	~50

Mocking practices: P5

- How often do you write tests?
 - Often, all the time
 - **Sometimes**
 - Rarely
- How often do you create / use mocks?
 - Often
 - Sometimes
 - **Rarely - would want to use mocks, esp. if they can be generated automatically**
- How familiar are you with Java + JUnit + Mockito?
 - Proficient
 - Comfortable
 - Novice
 - **N/A: I work with C#, .NET core, C++ game code**

The background is a faded, low-contrast image of a building with a series of large, rounded arches. To the right, a red traffic light is visible, and the overall scene appears to be an outdoor urban or institutional setting.

Thanks!

deepikat@kth.se