Software Engineering Essentials

Mock Object Pattern

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

Learning Goals

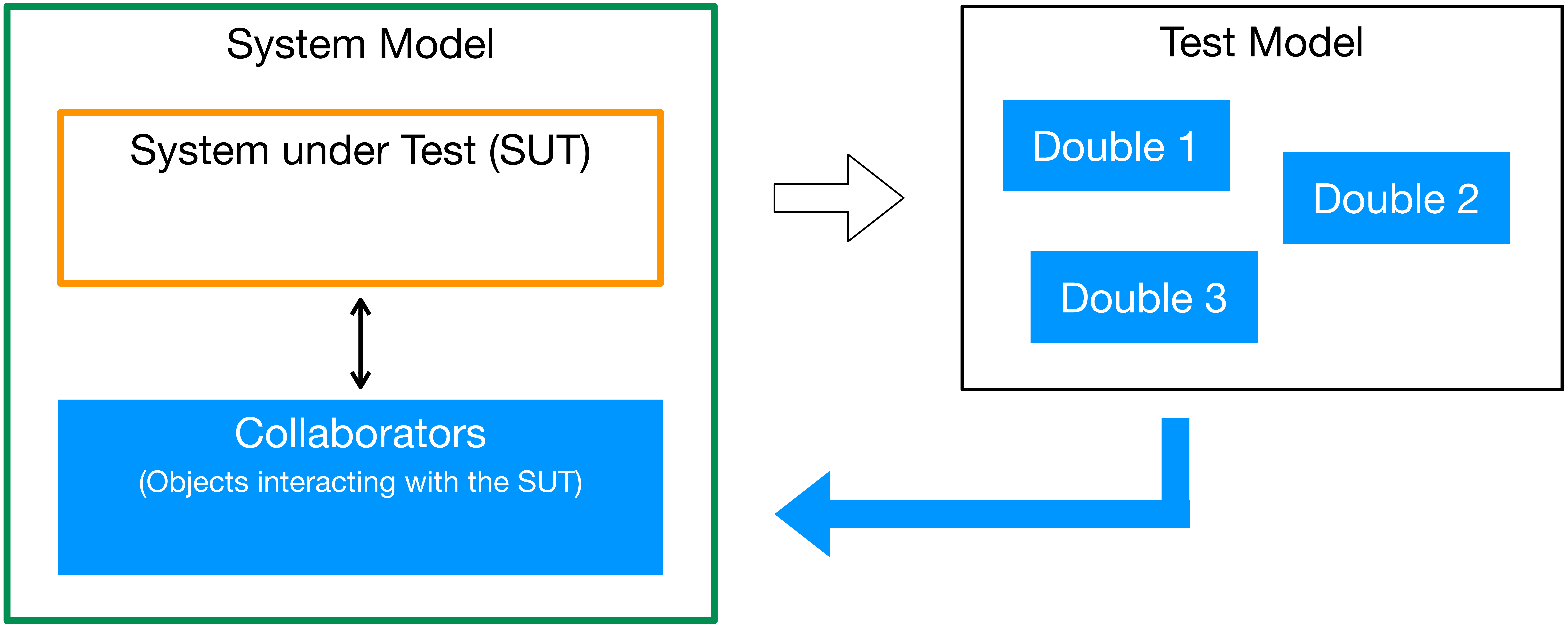
1. Understand object-oriented testing
2. Apply the mock object pattern

From State Testing to Behavior Testing

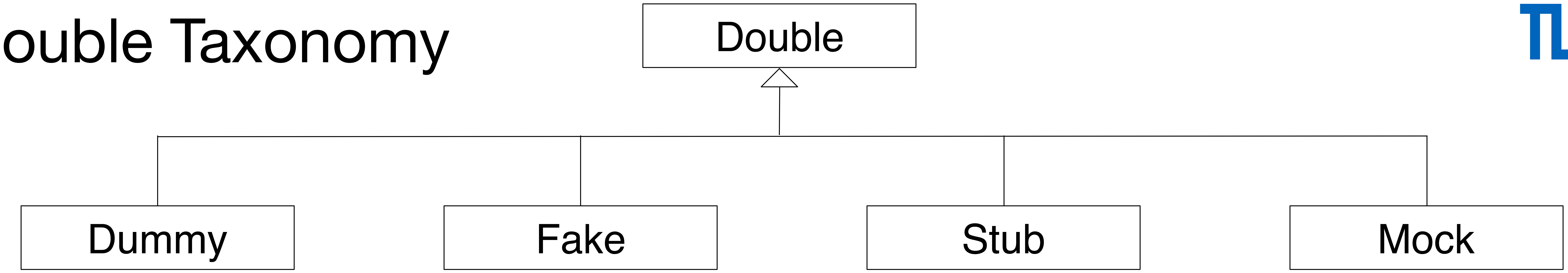
- Observation: Unit tests help us to to test the state of a SUT
- What if we want to test the state of a SUT, but also its interaction with the other components of the system, for example the interaction between Student and Course?
- Limitation of unit tests: you can not tests unit in isolations

—> Object-oriented testing and mock objects come into play

Object-Oriented Model-Based Testing



Double Taxonomy



Dummy

Often used to fill parameters lists, passed around but never actually used

Fake

A working implementation that contains a “shortcut” which makes it not suitable for production code

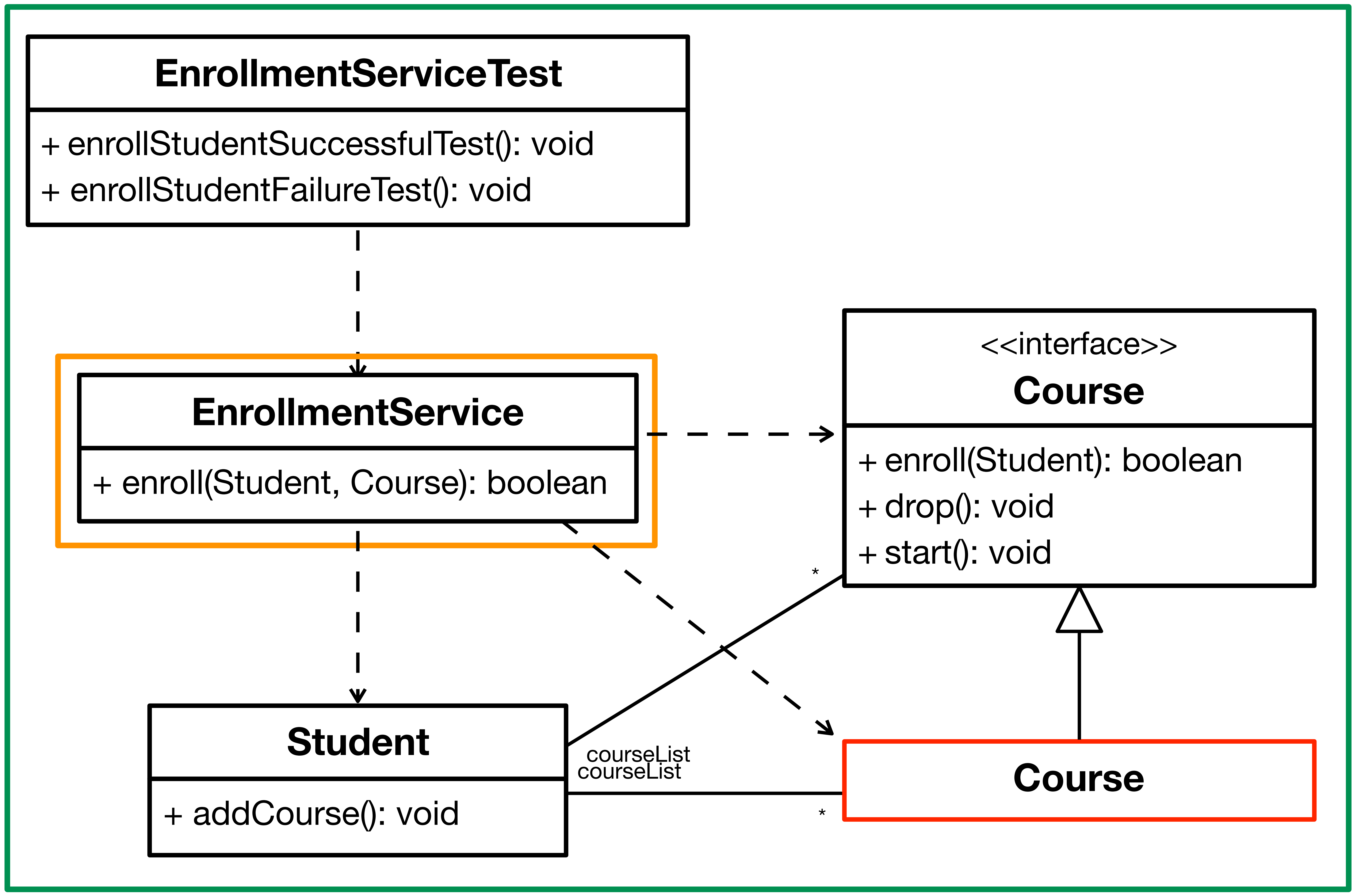
Stub

Provides canned answers to calls made during the test. Provides always the same answer

Mock

Mock objects are able to mimic the behavior of the real subject. They know how to deal with a specific sequence of calls they are expected to receive

Example University App (Motivation Mock Object)

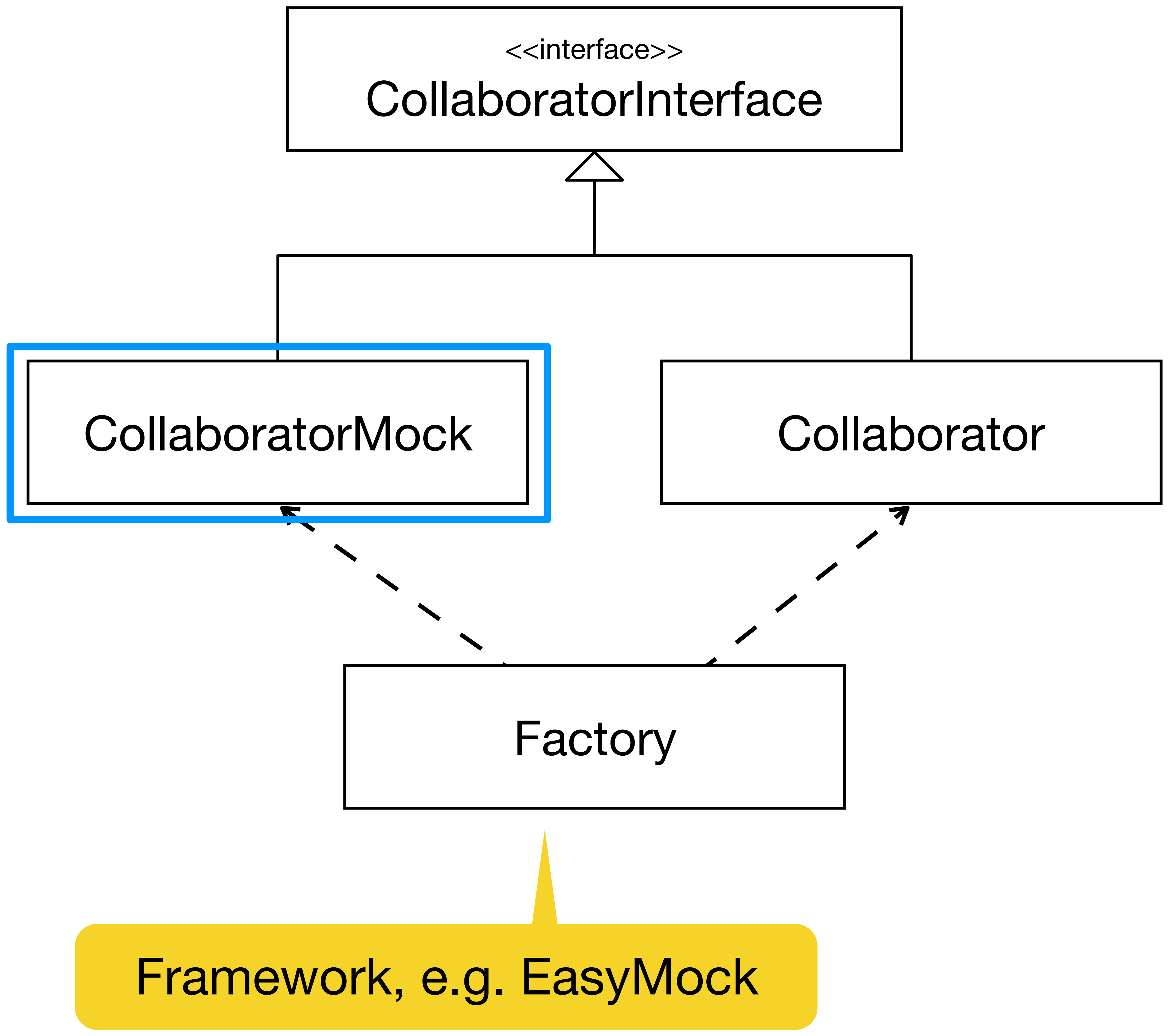


🟩 System model

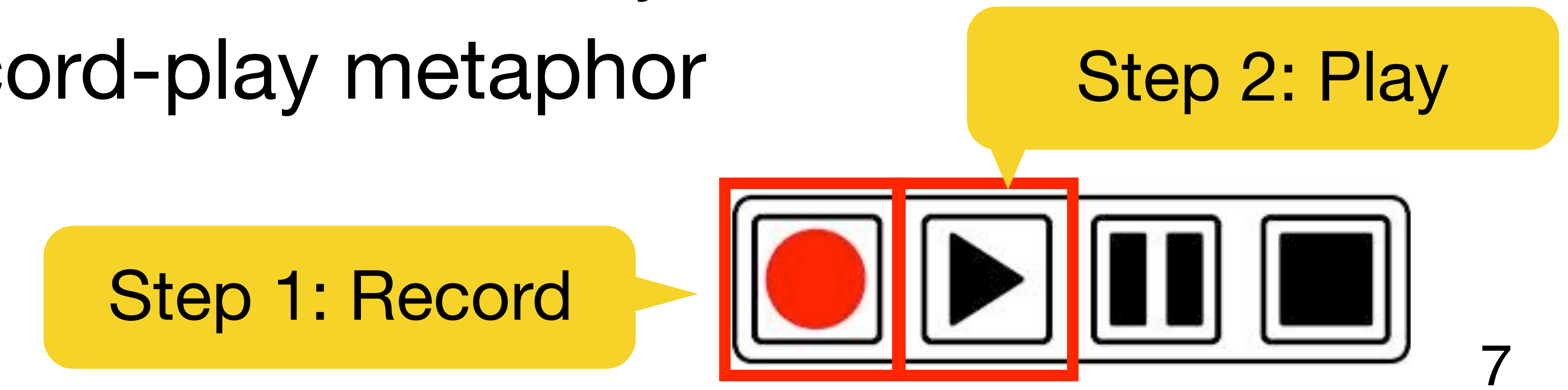
🟧 SUT

🟦 Test model

Mock Object Pattern



- In the mock object pattern a mock object replaces the behavior of a real object called the collaborator and returns defined values
- A mock object can be created at startup-time with the factory pattern
- Mock objects can be used for testing state of individual objects, as well as interaction between objects
- The use of mock objects is based on the record-play metaphor

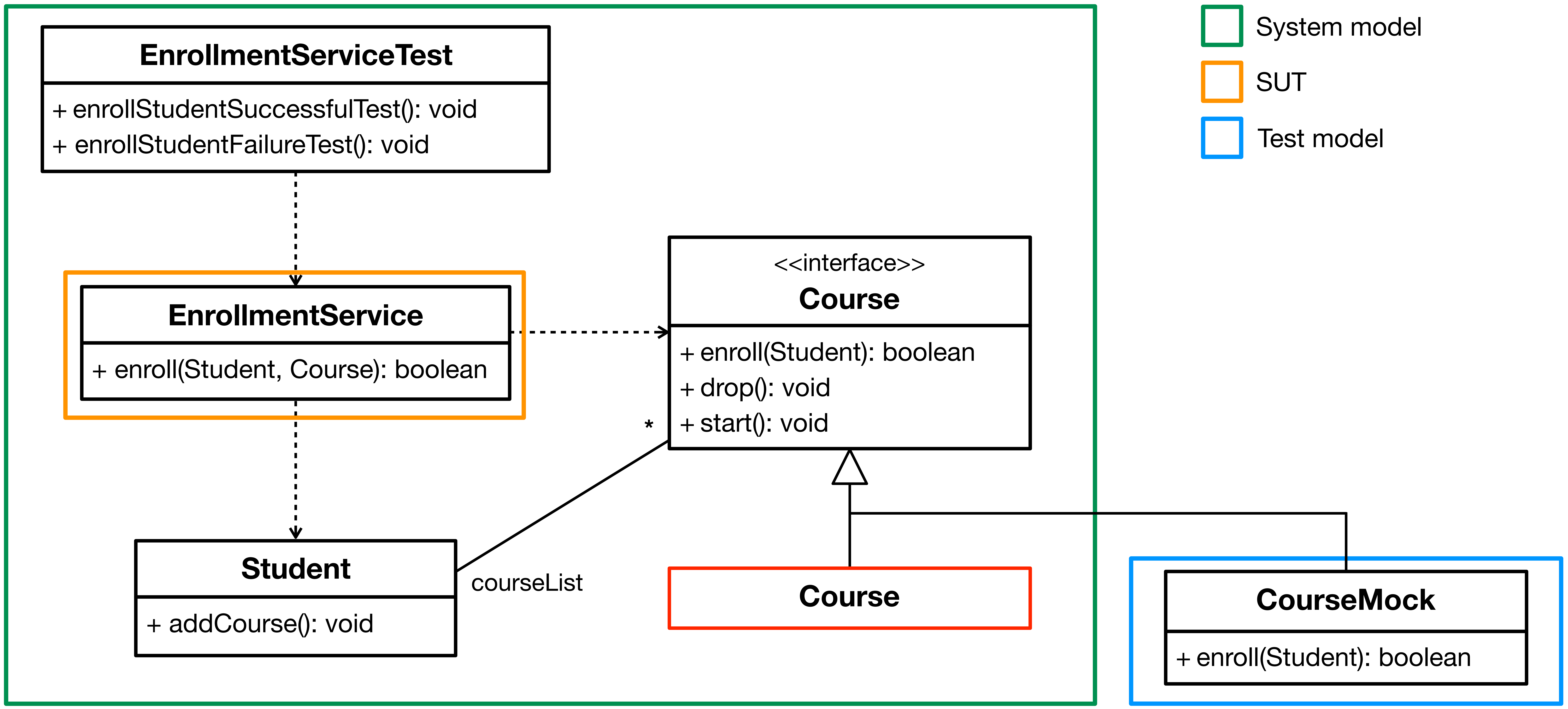


Record-Replay Metaphor for Mock Objects

5. Compare observed with expected behavior
4. Execute the SUT
Choir
Musical
Orchestra
2. Specify the expected behavior
1. Create the mock object
3. Make the mock object ready to play

🟩 System model
🟧 SUT
🟦 Test model

University App with a Mock Object


EnrollmentServiceTest
+ enrollStudentSuccessfulTest(): void
+ enrollStudentFailureTest(): void
EnrollmentService
+ enroll(Student, Course): boolean
<<interface>>
Course
+ enroll(Student): boolean
+ drop(): void
+ start(): void
*
Student
+ addCourse(): void
courseList
Course
CourseMock
+ enroll(Student): boolean

🟩 System model

🟧 SUT

🟦 Test model

Introduction EASYMOCK

- Open source testing framework for Java
- Annotations are used for test subjects (=SUT) and mocks

```
@TestSubject
private ClassUnderTest classUnderTest = new ClassUnderTest();

@Mock
private Collaborator mock;
```

- Specification of the behavior

```
expect(mock.foo(parameter)).andReturn(42);
```

- Make the mock ready to play

```
replay(mock);
```

- Documentation: http://easymock.org/user-guide.html

Unit test for Enrolling Students

```
@RunWith(EasyMockRunner.class)
public class EnrollmentServiceTest {

  @TestSubject
  private EnrollmentService enrollmentService = new EnrollmentService();

  @Mock
  private Course courseMock;

  @Test
  public void enrollStudentSuccessfulTest() {

    Student student = new Student("Andreas", "Seitz");
    int initialSize = student.courseList.size();

    expect(courseMock.enroll(student)).andReturn(true);
    replay(courseMock);

    enrollmentService.enroll(student, courseMock);

    assertEquals(initialSize + 1, student.courseList.size());
  }
}
```

1. Create the mock object

2. Specify the expected behavior

3. Make the mock object ready to play

4. Execute the SUT

5. Compare observed with expected behavior

From State Testing to Behavior Testing

- Observation: Mock objects help to test behavior
- Limitation of mock objects:
  - Mock objects might lead to high coupling between SUT and the rest of the system model

We would like to reduce this coupling as much as possible

- Dependency injection comes into play

More in the unit on **Dependency Injection**

Software Engineering Essentials

Mock Object Pattern

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics