Test Execution

Modulo 2.2

Capitolo 17 del libro

Learning objectives

- Appreciate the purpose of test automation
  - Factoring repetitive, mechanical tasks from creative, human design tasks in testing
- Recognize main kinds and components of test **scaffolding**
- Understand some key dimensions in test automation design
  - Design for testability: Controllability and observability
  - Degrees of generality in drivers and stubs
  - Comparison-based oracles and self-checks

Automating Test Execution

- Designing test cases and test suites is creative
  - Like any design activity: A demanding intellectual activity, requiring human judgment
- Executing test cases should be automatic
  - Design once, execute many times
- Test automation separates the creative human process from the mechanical process of test execution

Generation: From Test Case Specifications to Test Cases

- Test design often yields test case specifications, rather than concrete data
  - Ex: “a large positive number”, not 420023
  - Ex: “a sorted sequence, length > 2”, not “Alpha, Beta, Chi, Omega”
- Other details for execution may be omitted
- Generation creates concrete, executable test cases from test case specifications

Example Tool Chain for Test Case Generation & Execution

- We could combine ...
  - A combinatorial test case generation (ctwdge) to create test data
    - Optional: Constraint-based data generator to “concretize” individual values, e.g., from “positive integer” to 42
  - DDSteps to convert from spreadsheet data to JUnit test cases
  - JUnit to execute concrete test cases
- Many other tool chains are possible ...
  - depending on application domain

Scaffolding

- Code produced to support development activities (especially testing)
  - Not part of the “product” as seen by the end user
  - May be temporary (like scaffolding in construction of buildings
- Includes
  - Test harnesses, drivers, and stubs

Scaffolding ...

- Test driver
  - A “main” program for running a test
    - May be produced before a “real” main program
    - Provides more control than the “real” main program
      - To driver program under test through test cases
- Test stubs
  - Substitute for called functions/methods/objects
- Test harness (imbragatura)
  - Substitutes for other parts of the deployed environment
    - Ex: Software simulation of a hardware device

Unit Testing - Esempio

Esempio

```
foo(int x2, int y2) {
   ……
   gig(x2+2);
   ……
   }
testFoo() {
   ……
   foo(x1+1, y1-1);
   // controllo
}
gig(int x3) {
……
}
```

→ `foo`: test unit

Metodo da testare

→ `testFoo`: test driver

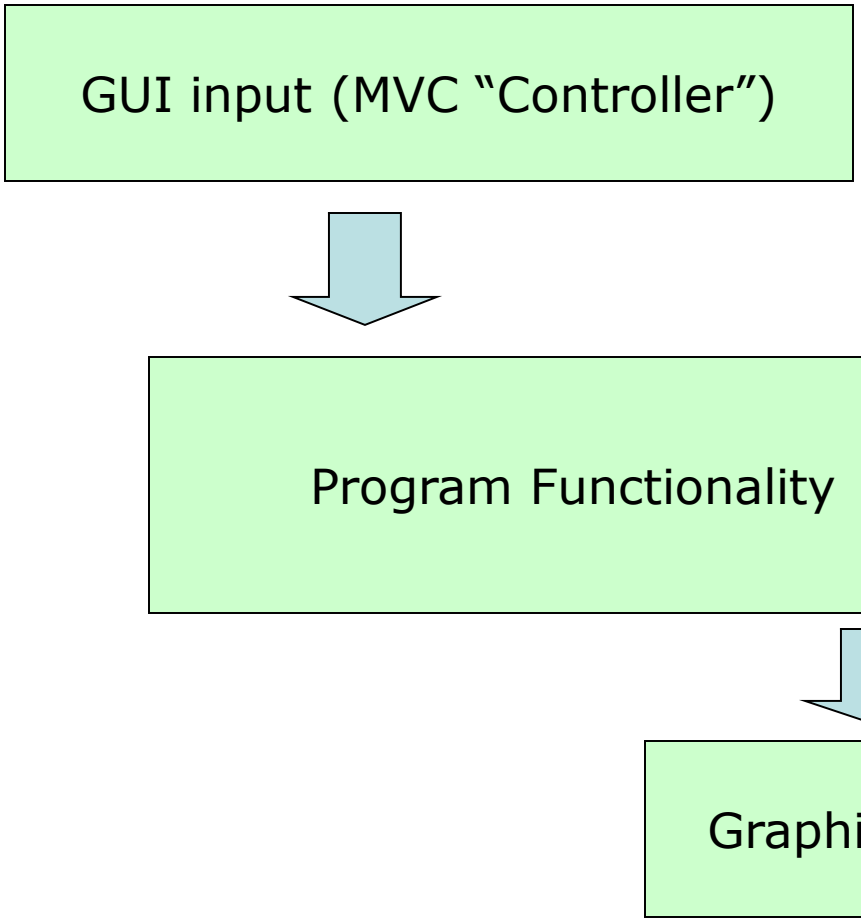
Metodo che testa foo

Simula una unità chiamante

→`gig`: test stub (opzionale)

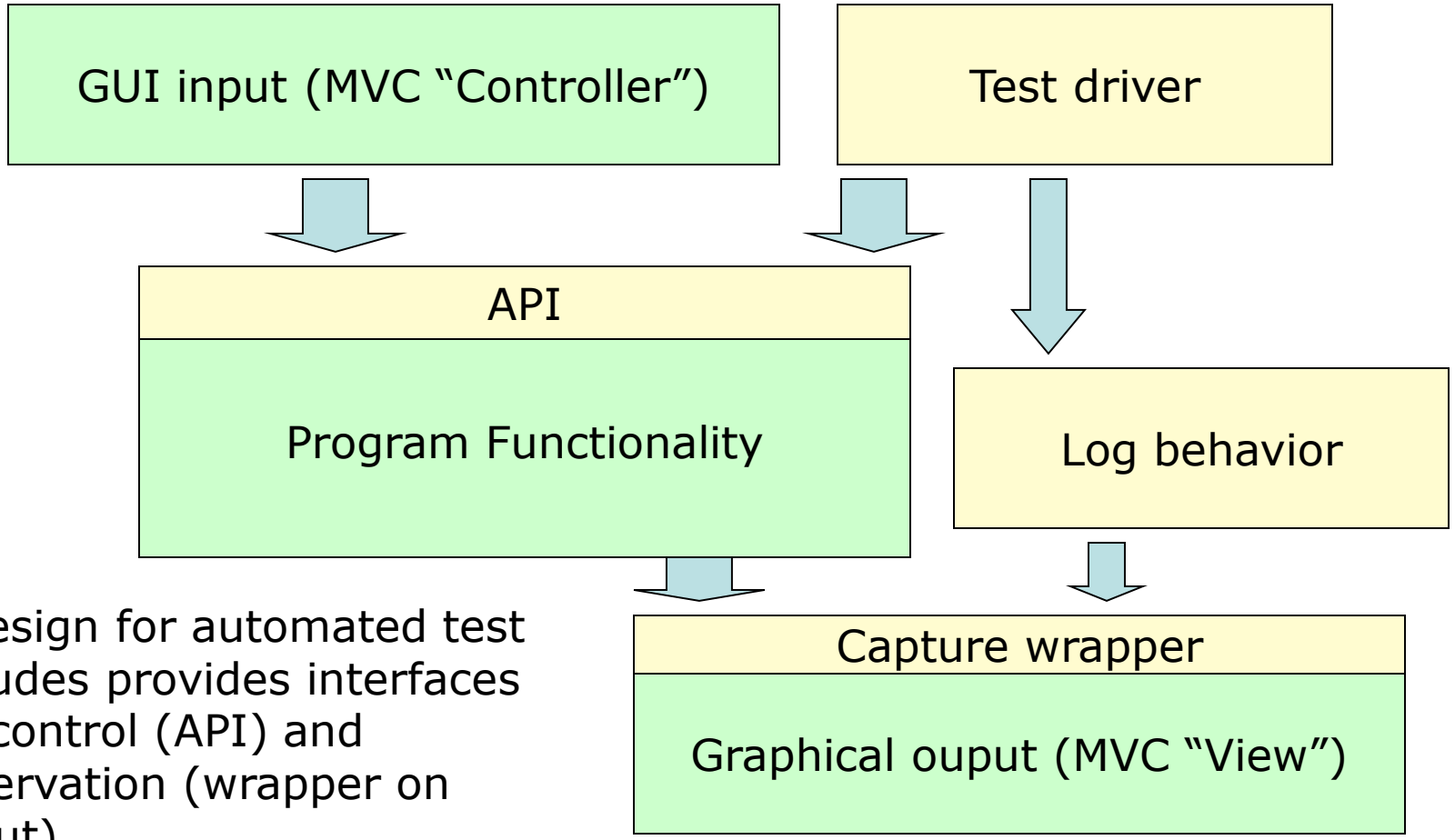
Simula un metodo chiamato da foo in modo di isolare il caso di test dal resto del sistema

Controllability & Observability



Example: We want automated tests, but interactive input provides limited control and graphical output provides limited observability

Controllability & Observability



A design for automated test includes provides interfaces for control (API) and observation (wrapper on ouput).

Generic or Specific?

- How general should scaffolding be?
  - We could build a driver and stubs for each test case
  - ... or at least factor out some common code of the driver and test management (e.g., JUnit)
  - ... or further factor out some common support code, to drive a large number of test cases from data (as in DDSteps)
  - ... or further, generate the data automatically from a more abstract model (e.g., network traffic model)
- A question of costs and re-use
  - Just as for other kinds of software

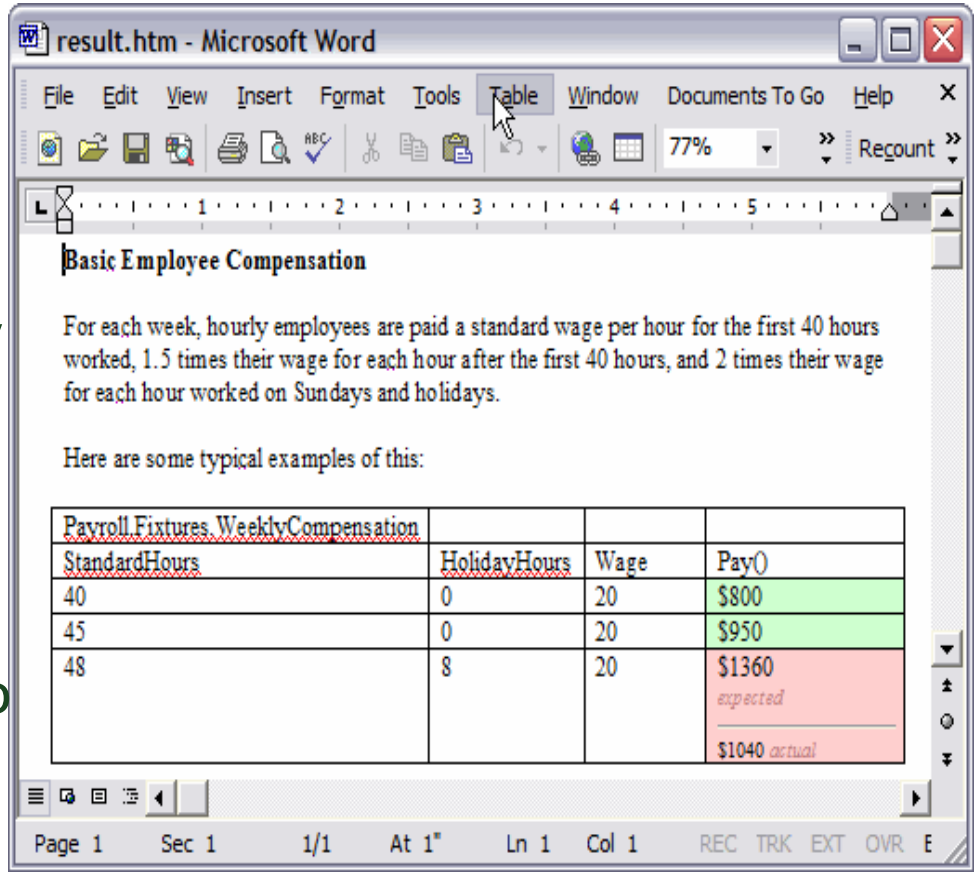
DDSTEPS

JUnit extension making test cases data driven. Uses external test data (in Excel, XML etc) which is injected into your test case using standard JavaBeans properties. Data enables and integrates toolkits such as jWebUnit and DbUnit. 100% JUnit compatible.

Example: FIT test

Fit works by reading tables in HTML files, produced with a tool like Microsoft Word. Each table is interpreted by a "fixture" that programmers write. The fixture checks the examples in the table by running the actual program.

In this example, the team is building a product to calculate employee pay. The team has worked together to create a Fit document that includes some examples of how hourly pay should be calculated.



Unit Testing - Esempio

→ `foo`: test unit

Metodo da testare

→ `testFoo`: test driver

Metodo che testa foo

Simula una unità chiamante

→`gig`: test stub (opzionale)

Simula un metodo chiamato da foo in modo di isolare il caso di test dal resto del sistema

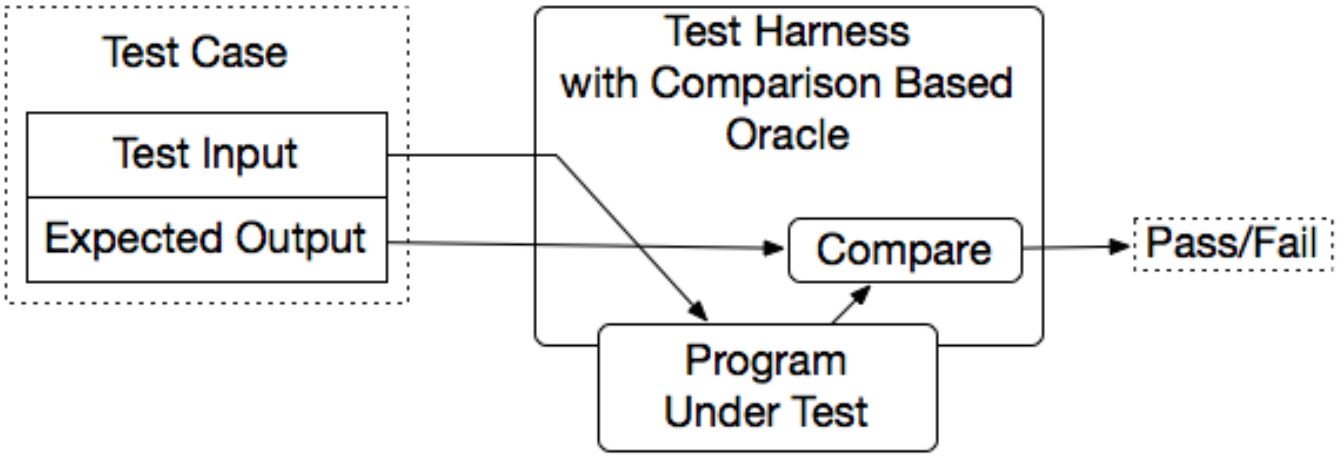
Esempio

```
foo(int x2, int y2) {
……
gig(x2+2);
……
}

testFoo() {
……
foo(x1+1, y1-1);
// controllo
}

gig(int x3) {
……
}
```

Oracles

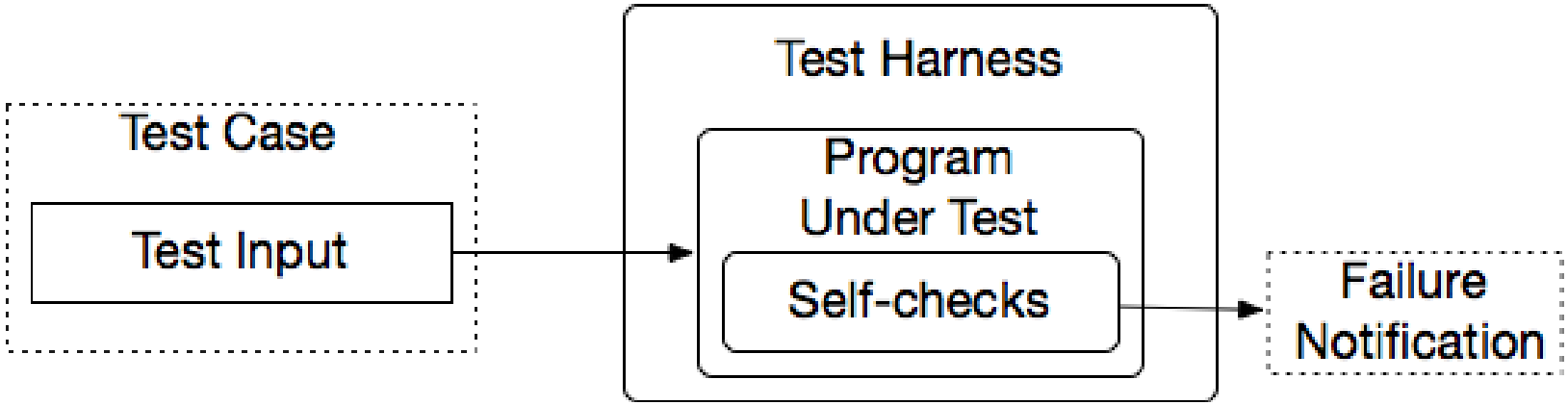
- Did this test case succeed, or fail?
  - No use running 10,000 test cases automatically if the results must be checked by hand!
- Range of specific to general, again
  - ex. JUnit: Specific oracle (“assert”) coded by hand in each test case
  - Typical approach: “comparison-based” oracle with predicted output value
  - Not the only approach!

Comparison-based oracle



- With a comparison-based oracle, we need predicted output for each input
  - Oracle compares actual to predicted output, and reports failure if they differ
- Fine for a small number of hand-generated test cases
  - E.g., for hand-written JUnit test cases

Self-Checking Code as Oracle



- An oracle can also be written as *self-checks*
  - Often possible to judge correctness without predicting results
- Advantages and limits: Usable with large, automatically generated test suites, but often only a *partial* check
  - e.g., structural invariants of data structures
  - recognize *many* or *most* failures, but not all

Oracle examples

Comparison-based

- Use assertion as in Junit
- assertEquals(5,y)

Self-Checking codes

- Use assertion in Java (part of the language)
- assert (s != null)
- ....

Using AssertJ

https://assertj.github.io/doc/

```
// entry point for all assertThat methods and utility methods (e.g. entry)
import static org.assertj.core.api.Assertions.*;

// basic assertions
assertThat(frodo.getName()).isEqualTo("Frodo");
assertThat(frodo).isNotEqualTo(sauron);

// chaining string specific assertions
assertThat(frodo.getName()).startsWith("Fro").endsWith("do").isEqualToIgnoring
Case("frodo");

// collection specific assertions (there are plenty more)
// in the examples below fellowshipOfTheRing is a List<TolkienCharacter>
assertThat(fellowshipOfTheRing).hasSize(9).contains(frodo,
sam).doesNotContain(sauron);

// as() is used to describe the test and will be shown before the error message
assertThat(frodo.getAge()).as("check %s's age",
frodo.getName()).isEqualTo(33);
```

Capture and Replay

- Sometimes there is no alternative to human input and observation
  - Even if we separate testing program functionality from GUI, some testing of the GUI is required
- We can at least cut *repetition* of human testing
- *Capture* a manually run test case, *replay* it automatically
  - with a comparison-based test oracle: behavior same as previously accepted behavior
    - reusable only until a program change invalidates it
    - lifetime depends on abstraction level of input and output

Esempio

- Record and playback any web application. Recording saves time and helps non-technical users contribute to automation.
- The Sahi Controller helps easily identify and experiment with elements on any browser.
- The same script works on all browsers.

```
_click(_link("Login"));
_setValue(_textbox("username"), $usr);
_setValue(_password("password"), $pwd);
_click(_submit("Login"));
```

Esempio con Selenium

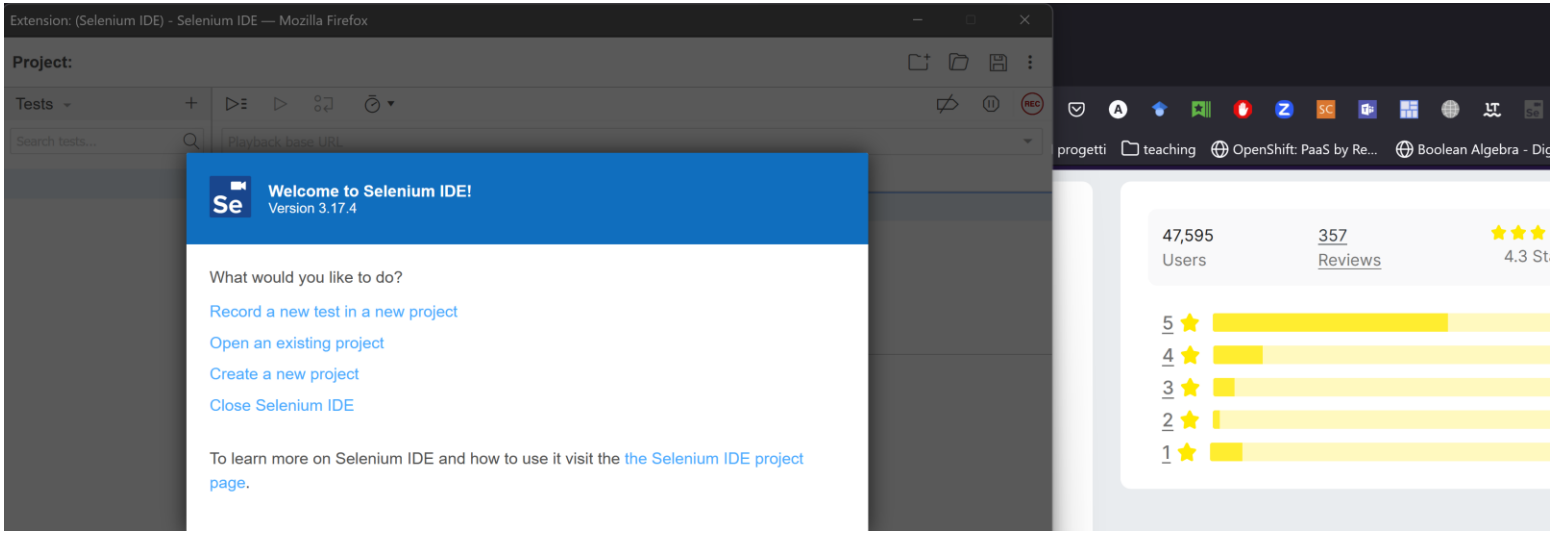
- Testing automatico di applicazioni web
- Usato per capture-replay
  - Selenium ide come plugin del browser
  - Gli script sono salvati in un formato testuale (linguaggio selenese) e possono essere modificati e reiseguiti
- Oppure posso scrivere dei test nei maggiori linguaggi di programmazione
  - Usando la libreria Selenium WebDriver
  - Simula l'uso del browser

1. capture and replay

- Demo con firefox

1) Installa plugin
2) Registra
3) Usa il browser
4) Salva lo script
5) rieseguilo

Running selenium



2. scrivere casi di test

- installo chrome e chrome driver (o per il TUO browser)
  - Selenium non riesce a controllare il browser direttamente
- costruisco un progetto java (ad esempio con maven)
  - aggiungo il jar di selenium
- scrivo il caso di test come @Test o come programmi
- metto i comandi per fare azioni nel browser e controllare che il tutto funzioni

Vedi codice su github progetto + demo

2. scrivere casi di test

- installo chrome e chrome driver (o per il TUO browser)
  - Selenium non riesce a controllare il browser direttamente
- costruisco un progetto java (ad esempio con maven)
  - aggiungo il jar di selenium
- scrivo il caso di test come @Test o come programmi
- metto i comandi per fare azioni nel browser e controllare che il tutto funzioni

Vedi codice su github progetto + demo

Mocking frameworks

mock objects are simulated objects that mimic the behavior of real objects in controlled ways

- A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.

Advantages of using mock objects

- when the real object are too difficult to use
  - e.g. a database
  - Slow, using networks, or other types of infrastructure
- When the real object have non deterministic behavior
  - And the test may fail for that
  - eg. Time, random …
- Before they exist
  - Not implemented yet

Mock in brief

In your unit tests, you want to test certain functionality (the class under test) in isolation. Other functionality required to test the class under test, should be controlled to avoid side-effects.

A mock object is a dummy implementation for an interface or a class. It allows to define the output of certain method calls. They typically record the interaction with the system and tests can validate that.

You can create mock objects manually (via code) or use a mock framework to simulate these classes. Mock frameworks allow you to create mock objects at runtime and define their behavior.

The classical example for a mock object is a **data provider**. In production an implementation to connect to the real data source is used. But for testing a mock object simulates the data source and ensures that the test conditions are always the same.

These mock objects can be provided to the class which is tested. Therefore, the class to be tested should avoid any hard dependency on external data.

Mocking or mock frameworks allows testing the expected interaction with the mock object. You can, for example, validate that only certain methods have been called on the mock object.

Using **mockito** to build stubs

Mockito can be used to “mock” real objects with simpler version (like DB):

```
//
LinkedList mockedList = mock(LinkedList.class);
// stubbing appears before the actual execution
when(mockedList.get(0)).thenReturn("first");
// the following prints "first"
System.out.println(mockedList.get(0));
// the following prints "null" because get(999) was not stubbed
System.out.println(mockedList.get(999));
```

Esempio più complesso

Ho una implementazione incompleta che voglio testare

Posso creare un oggetto mock e

- specificare cosa fare quando un certo metodo viene chiamato

- verificare che il metodo venga chiamato

- controllare il risultato

Per applicazioni mobili?

http://appium.io/
Appium is an open source test automation framework for use with native, hybrid and mobile web apps.
It drives iOS, Android, and Windows apps using the WebDriver protocol.

Applicazioni GUI

Summary

- Goal: Separate creative task of test design from mechanical task of test execution
  - Enable generation and execution of large test suites
  - Re-execute test suites frequently (e.g., nightly or after each program change)
- Scaffolding: Code to support development and testing
  - Test drivers, stubs, harness, including oracles
  - Ranging from individual, hand-written test case drivers to automatic generation and testing of large test suites
  - Capture/replay where human interaction is required