

Code comprehension

COMP23420: Software Engineering

Week 3

Markel Vigo

Week

Course Unit Roadmap (Weeks 2-10)

Skills for Small Code Changes Skills for Adding Features Working with source code repositories **Estimating for** Debug software change **Test** Software Coding defensively Code reading architecture Code review Domain specific Design for testability languages Safe migration of functionality 2 3 4 5 6 8 10

Link to the Coursework/Exam

- We will learn how to understand a unfamiliar domain and codebase
- We will put in practice reading strategies
- We will use the search functionalities provided by the environment to find the location of a specific piece of code and bugs
- We will learn how to read unit tests to understand the codebase
- We will learn how to write unit tests to understand the codebase

Outline

- 1. Motivation 5'
- 2. Learning unfamiliar codebases 25'
- 3. Unit testing overview 25'
- 4. Break 10'
- 5. Unit test reading 15'
- 6. Unit test writing 30'

Motivation

```
▼ A marauroa

▼ A client

   ▶ Æ net
   ▶ DannedAddressException.java
   ▶ III Client.java
   ▶ D LoginFailedException.java
   package-info.java
   ▶ In TimeoutException.iava
  ▼ Æ common
   ► Æ crypto
   ▶ Æ game
   ▶ 🌐 io
   ▶ Æ net
   ConfigurationParams.iava
   ▶ J Log4J.java
   ▶ Dogger.java
   ▶ MarauroaUncaughtExceptionHandler.java
   package-info.java
   Pair.java
   ▶ In TimeoutConf.java
   Utility.java
  ▼ Æ server
   ▶ 🔠 db
   ▶ Æ game
   ▶ Æ net
   ▼ 🕞 marauroad

§ logger

          narauroa
          ₹ VERSION
          s main(String[]): void
          setArguments(String[]): void
          gameMan
          netMan
          □ rpMan
          marauroad()
          createBeanForStatistics(): void
          finish(): void
          getVersionMai(): int
          getVersionMin(): int
```

```
Hash.random(4);
}.start();
try {
    netMan = new marauroa.server.net.nio.NIONetworkServerManager();
    netMan.start();
} catch (Exception e) {
    logger.error("Margurog can't create NetworkServerManager.\n" + "Reasons:\n"
            + "- You are already running a copy of Marauroa on the same TCP port\n"
            + "- You haven't specified a valid configuration file\n"
            + "- You haven't create database\n"
            + "- You have invalid username and password to connect to database\n", e);
    return false;
}
    rpMan = new RPServerManager(netMan);
    rpMan.start();
} catch (Exception e) {
    logger.error(
                     "Marauroa can't create RPServerManager.\n"
                            + "Reasons:\n"
                            + "- You haven't specified a valid configuration file\n"
                            + "- You haven't correctly filled the values related to game configuration.
                            + "- There may be an error in the Game startup method.\n", e);
    return false;
}
    if (Configuration.getConfiguration().get("n") == null) {
        throw new Exception("Missing RSA key pair in server.ini; run marauroa.tools.GenerateKeys");
    RSAKey key = new RSAKey(new BigInteger(Configuration.getConfiguration().get("n")),
            new BigInteger(Configuration.getConfiguration().get("d")), new BigInteger(
                    Configuration.getConfiguration().get("e")));
    gameMan = new GameServerManager(key, netMan, rpMan);
    gameMan.start();
} catch (Exception e) {
    logger.error(
                     "Marauroa can't create GameServerManager.\n"
                            + "Reasons:\n"
                            + "- You haven't specified a valid configuration file\n"
                            + "- You haven't correctly filled the values related to server information
                    e);
    return false;
}
```



Motivation

- What activity takes up most of a maintenance programmer's time?
- Where can we get information about a software system from?
- When we need to get information about a software system?

Learning unfamiliar codebases

```
▼ Æ client
  ▶ Æ net
  ▶ ■ BannedAddressException.iava
  ▶ II Client.java
  ▶ ☐ ClientFramework.java
  LoginFailedException.java
 package-info.java
  ► In TimeoutException.java
▼ Æ common
  ► Æ crypto
  ▶ Æ game

→ io

  ▶ Æ net
  ► D Configuration.java
  ConfigurationParams.java
  ▶ 🚺 Log4J.java
  ▶ Dogger.java
  ▶ MarauroaUncaughtExceptionHandler.java
  Dackage-info.java
  Pair.java
  ▶ In TimeoutConf.iava
  Utility.java
▼ Æ server
  ▶ 🔠 db
  ▶ Æ game
  ▶ Æ net

§F logger

         narauroa
         ₹ VERSION
         S getMarauroa(): marauroad
         of main(String[]): void
         setArguments(String[]): void
         gameMan
         netMan
         □ rpMan
         marauroad()
         createBeanForStatistics(): void
         finish(): void
         getVersionMai(): int
         getVersionMin(): int
```

```
Hash.random(4);
}.start();
    netMan = new marauroa.server.net.nio.NIONetworkServerManager();
    netMan.start();
    logger.error("Marauroa can't create NetworkServerManager.\n" + "Reasons:\n"
            + "- You are already running a copy of Marauroa on the same TCP port\n"
            + "- You haven't specified a valid configuration file\n"
            + "- You haven't create database\n'
            + "- You have invalid username and password to connect to database\n", e);
    return false;
    rpMan = new RPServerManager(netMan):
    rpMan.start();
} catch (Exception e) {
    loager.error(
                     'Marauroa can't create RPServerManager.\n"
                            + "Reasons:\n"
                             + "- You haven't specified a valid configuration file\n"
                            + "- You haven't correctly filled the values related to game configuration.
                            + "- There may be an error in the Game startup method.\n", e);
    return false;
    if (Configuration.getConfiguration().get("n") == null) {
        throw new Exception("Missing RSA key pair in server ini; run marauroa.tools.GenerateKeys");
    RSAKey key = new RSAKey(new BiaInteger(Configuration.getConfiguration().get("n")),
            new BigInteger(Configuration.getConfiguration().get("d")), new BigInteger(
                    Configuration.getConfiguration().get("e")));
    qameMan = new GameServerManager(key, netMan, rpMan);
    gameMan.start();
} catch (Exception e)
    logger.error(
                     Marauroa can't create GameServerManager.\n"
                            + "Reasons:\n"
                            + "- You haven't specified a valid configuration file\n"
                             + "- You haven't correctly filled the values related to server information
    return false;
```



Code reading

- Goal: build a mental model of the codebase by acquiring
 - Application domain knowledge
 - General programming knowledge
- Use systematic reading strategies
 - Top-down
 - Bottom-up
- Tip: assume previous coders were sensible and honest

Comprehending Marauroa

- What do we know about the domain?
 - Marauroa is a game engine
 - It's used to develop MMORPGs
 - Massive Multiplayer Online Role-Playing Videogame





Comprehending Marauroa: get an overview

- Top-down reading strategies at the outset:
 - Take advantage of the hierarchical structure of the codebase
 - Ignore unimportant and distracting details
 - Expand the tree and read the names of packages
- What hypotheses can we formulate by doing just this:?
 - There are three main packages named client,
 server and common → client/server architecture
 - There is some sort of network communication and there is some messaging (probably between the client and the server)
 - There is a database



Comprehending Marauroa: find key classes

- Follow the gradual expansion of the hierarchy and read class names
 - At higher levels of the hierarchy
 - With meaningful names
- Identify the classes that play a central role
 - By opening and skimming over these classes
 - Check the import clause to see their dependencies
- Check for instance marauroad.java within the server package

Comprehending Marauroa: reading classes

- Classes can also be read gradually using a top-down strategy
 - Look at the icons provided by the environment
 - Skim the comments use Javadoc
 - Skim the attributes and methods
- Don't need to understand every word to understand overall meaning
- Which hypotheses can we establish about marauroad.java class?
 - It's a daemon running on the server side
 - It's a thread that throws 4 threads
 - It follows a singleton pattern



Use the search functionalities of the IDE

- Find the classes of the threads thrown by marauroad.java
- Global search: the IDE offers different options
 - 1. You can navigate the hierarchy. The path pops-up when hovering the class.
 - 2. Use the search functionality of the menu
 - 3. Use the mouse by
 - Select the text
 - ii. Right key mouse click
 - iii. Select references/declarations
 - iv. Select the scope
 - v. Select the class from the results

Learning unfamiliar codebases

- Acquire knowledge gradually through the top-down reading strategy
- Packages and classes are hierarchical
- Location is indicator of relevance
- Extract meaning from names
- Open files and skim
- Use different sources
- Don't read everything
- Trust the previous developer



Unit testing overview

```
▼ ⊞ common
   ▼ ∰ game
     BugAtApplyDifferencesTest.java
     ► I GetAndApplyDifferencesTest.java
      ▶ I RPClassTest.iava
     RPClassTestHelper.java
      ▶ I RPEventTest.java
      ► I RPObjectDelta2Test.java
      ▶ I RPObjectIDTest.java
      RPObjectTest.java
     RPSlotTest.java
   ▶ 册 io
   ▶ ⊞ net
  ▼ ∰ server
   ▶ 册 db

▼ ∰ game

       ▶ I) PlayerEntryContainerTest.java
       ▶ 🔠 db
   ▶ 册 net
  ► IniFileExistsTest.java
```

```
* @author hendrik
public class CharacternameValidationTest {
    private static Logger logger = Logger.getLogger(CharacternameValidationTest.class);
    public boolean checkImpersonation(String username) {
        String name = username:
        name = name.replaceAll("[ _..;.\\-\\\ \"�$%&/()='<>|*+~#]", " ");
        if (name.startsWith(" ") || name.endsWith(" ") || (name.indexOf("gm ") > -1) || (name.indexOf(" gm") > -1)
                | | name.startsWith("gm") | | name.endsWith("gm") | | (name.indexOf(" ") > -1)) {
            logger.warn("Possible impersonation: " + username);
            return false;
        return true:
    public void checkImpersonation() {
        Assert.assertTrue(checkImpersonation("hendrik"));
        Assert.assertTrue(checkImpersonation("hendrik1"));
        Assert.assertTrue(checkImpersonation("Hendrik"));
        Assert.assertTrue(checkImpersonation("hEndrik1"));
        Assert.assertFalse(checkImpersonation("hendrik_"));
        Assert.assertFalse(checkImpersonation("_hendrik"));
        Assert.assertFalse(checkImpersonation("hendrikgm"));
```

Unit testing: what, why & when?

- Replicating problems is key to bug fixing
- What about the debugger and System.out.println?
- Marauroa is a game engine
 - We need to build a videogame to try it's functionalities!
 - Even if we build a toy game it could take ages to replicate bug
- Unit testing is an elegant and maintainable way of executing pieces of code
- Test-driven development lifecycle:



Unit testing: terminology

- Terminology
 - Test fixture
 - Code under test
 - Unit test
 - Test coverage
 - Integration test
- Organisation
 - Where deploy tests?
 - Which part of the code to test?



JUnit annotations

- @Test indicates the method as a test method
- @Before to execute a method before a test
- @BeforeClass to execute a method before all tests
- @After to execute a method after a test
- @AfterClass to execute a method before all test

JUnit statements

```
assertTrue(String message, boolean condition)
Checks that the boolean condition is true.
```

```
assertFalse(String message, boolean condition)
Checks that the boolean condition is false.
```

```
assertEquals(String message, expected, actual)
Tests that two values are the same.
```

```
assertNull(String message, object)
Checks that the object is null.
```

```
assertNotNull(String message, object)
Checks that the object is not null.
```



Let's try it ourselves

- Create a new project with two packages
- Code under test contains a class with two methods
 - 1. Sums two integers
 - 2. Links two strings
- Another package contains a class with tests to check all is working properly

What do you need to use:

@Before to execute a method before a test

@Test indicates the method as a test method

assertEquals(String message, expected, actual) tests that two values are the same

One Possible Solution

Code under test

```
package source;
public class CodeUnderTest {

public int addInteger (int a, int b){
    return a+b;
}

public String linkStrings (String a, String b){
    return a+b;
}
}
```

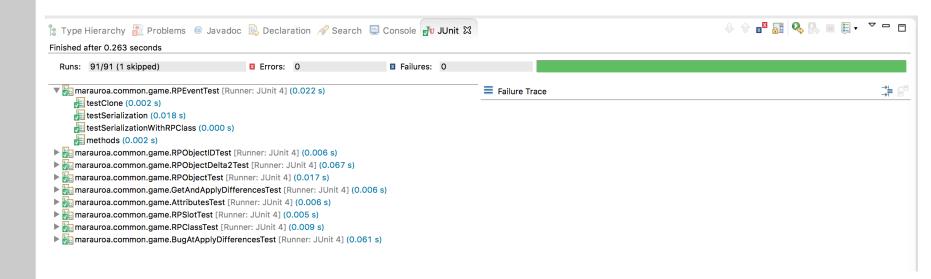
Unit test code

```
package tests;
import static org.junit.Assert.*;
 import org.junit.Before;
 import org.junit.Test;
 import source.CodeUnderTest;
 public class TestsforCodeUnderTest {
     private CodeUnderTest cut;
     @Before
     public void buildFixture(){
         cut=new CodeUnderTest();
     @Test
     public void sumTest() {
          assertEquals("I'm expecting 18",18, cut.addInteger(7, 11));
     @Test
     public void linkTest() {
          assertEquals("I'm expecting 711","711", cut.linkStrings("7","11"));
```

Break: 10 minutes



Unit test reading & writing





Unit testing: terminology

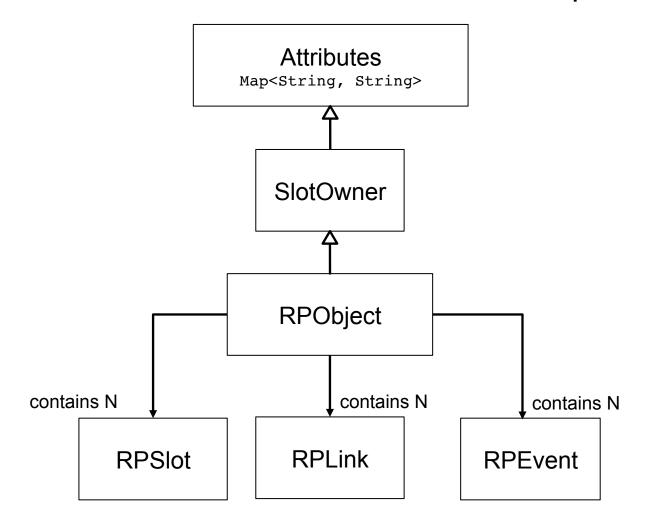
- Terminology
 - Test fixture
 - Code/system under test
 - Unit test
 - Test coverage
 - Integration test
- Organisation
 - Where deploy tests?
 - Which part of the code to test?

What can we learn from unit test reading?

- Unit test indicate traces of
 - Critical and key pieces of code
 - Tricky parts of the code
 - Sensitive excerpts
 - Bugs that were repaired in the past
- Marauroa has a dedicated package with tests
- Check marauroa.common.RPObjectTest.java
- What can we learn about the class being tested?
 - Star reading the method under the @Before clause
 - How is it structured and which are its dependencies?
 - Which is the main data structure?

What can we learn from unit test reading?

How is it structured and which are its dependencies



Testing worlds, zones and objects

- There is one world which can contain N zones and zones contain objects
- Find the classes in Marauroa

Object: marauroa.common.game.RPObject

Zone: marauroa.server.game.rp.MarauroaRPZone

World: marauroa.server.game.rp.RPWorld

- Let's test whether
 - 1. There is only one instance of World
 - 2. If Zones are actually added to Worlds
 - 3. If Objects are actually added to Zones
 - 4. If Objects are destroyed when removed from Zones



Testing worlds, zones and objects

- 1. Test there is only one instance of World
 - Get two instances of World
 - ii. Use a JUnit a statement to compare the two instances
- 2. Test if Zones are actually added to Worlds
 - Get an instance of the world
 - ii. Create a new Zone
 - iii. Add the new Zone to the World
 - iv. Use a method from the World class to check if our Zone belongs to the World
 - v. Use a JUnit a statement to check the above



Testing worlds, zones and objects

- 3. Test Objects are actually added to Zones
 - i. Create a Zone and create and Object
 - ii. Set an identifier to the Object
 - iii. Add the object to Zone
 - iv. Use a method from the Zone class to check if our Object belongs to the Zone.
 - v. Use a JUnit a statement to check the above
- 4. Test Objects are destroyed once they are removed
 - Same as above until step v.
 - ii. Remove object from zone
 - iii. Use a method from the Zone class to check if our Object belongs to the Zone. Use JUnit.



Next Week

 In the team study sessions will keep fixing bugs and using. Commit modifications.

- In the workshop we will learn to
 - Do code reviews
 - Use Git workflows