Universität Zürich[UZH]

RERG

Software Quality Exercise 2

Testing and Debugging

1 Information

1.1 Dates

- Release: 12.03.2012 12.15pm
- Deadline: 19.03.2012 12.15pm
- Discussion: 26.03.2012

1.2 Formalities

Please submit your solution as a *pdf* and submit it via email to `charrada@ifi.uzh.ch`. The subject of the email must begin with *[FS 12 SWQ]*. Exercises should be solved and handed in in groups of three. Every member of a group must be able to answer questions about the group's solution. The document must include the names of group members.

1.3 Daiquiri

To support this exercise, we have created a *trac environment* on *Daiquiri*, one of our internal servers[1]: `http://daiquiri.ifi.uzh.ch/trac/swq12/`. Trac provides a wiki, on which you will find pointers to the documentation of the various tools used in this exercise.

Registration

To perform Exercise 2 of this assignment you require an account to access to the version control repository. As obtaining an account **requires the intervention of the assistant**, we strongly recommend you to ask for it **as early as possible**.

[1] To access it, you need to be connected to the university network, either physically (in lab rooms or via wifi) or through a VPN connection.

Every participant (even if you work in a group) has to perform the following steps:

a) Register on this website: `https://daiquiri.ifi.uzh.ch/trac/swq12/register`. As username, choose the *sxxyyyzz* (where *xxyyyzz* is your student number without the last digit) part of your student email address. Do not forget to enter a valid email address, in order to be notified of any updates on your tickets.
b) Create a new ticket to ask for access to the version control repository. Fill in the form with as much detail as possible. Assign it to the assistant: *m1053679*. Write down the number of your ticket.
c) Wait on the assistant to confirm your access with a comment on your ticket.

1.4 Tools

For this exercise, you will use various tools used to develop software with an evolutionnary process. More precisely, you will need an IDE for Java programs (such as Eclipse or Netbeans), *SVN* and *Maven (2 or 3)*. You can find out more about these tools on the wiki hosted on *daiquiri*.

2 Introducing ImageJ

This exercise is based on an existing software: *ImageJ*. ImageJ is an image processing tool written in Java. With the help of plugins and macros, it can be extended with new features, such as the support of new file formats or new analysis. Still, the current architecture of ImageJ has reached its limits in terms of extensibility: many requested features (such as the support of dynamic charts or the ability to use ImageJ on a cluster of computers) cannot be implemented unless the tool is deeply refactored.

2.1 Getting ImageJ

The source code of ImageJ is hosted on an SVN repository located on *Daiquiri*.

a) With a browser, visit the following URL:
`https://daiquiri.ifi.uzh.ch/svnswq12/imagej/`
Provide the *username* and the *password* you have registered within *Trac*.
b) In an SVN repository, a project is layouted in 3 directories: `trunk`, `tags` and `branches`. What are these directories used for?
c) Check out the trunk of the project with the following command:

```
svn co https://daiquiri.ifi.uzh.ch/svnswq12/imagej/trunk/ ImageJ
```

The error message is due to the fact that the server certificate has not been signed by a CA. Still, accept it *permanently*. Again, provide the username and the password you have entered in Trac.

2.2 Building ImageJ

There is a *pom.xml* file in the `ImageJ` directory. This file is used by Maven to build ImageJ, that is, to compile its source code, to execute its test suite and to package it into a jar file.

a) Within the `ImageJ` directory, execute the following command:

```
mvn install
```

Was the build successful?

b) Present briefly the content of the *pom.xml* file and the standard layout of a Maven project.

Note that once the build is complete, you can execute ImageJ by using the *jar* file built by Maven (`java -jar target/ImageJ-1.4.3q.jar`).

2.3 Working on ImageJ

In the repository, there is no metadata about the project for any IDE. Nevertheless, Netbeans can import Maven projects while Maven can create an Eclipse project (with the command `mvn eclipse:eclipse`). Note that before importing such a project in Eclipse, you need to setup the classpath variable M2_REPO (Window, Preferences, Java, Build Path, Classpath Variables) to the repository of Maven2 (which is usually located in `$home/.m2/repository`, where `$home` is your home directory). After you have imported the project in an IDE, you can launch ImageJ by executing its main class whose name is `ImageJ`.

One of the problem of ImageJ's current architecture is that ImageJ data models and image processing algorithms are tightly coupled to the GUI. Find 2 examples in the code where a single object both transforms images and displays something in the GUI. Discuss, in maximum 10 sentences, why this problem impacts both the extensibility and the testability of ImageJ.

2.4 Improving the source code of ImageJ

In Trac, there are 10 "enhancement" tasks available. Pick one of them (1 per group) and mark it as accepted (so that no assignment is solved twice). Each assignment concerns one class. Assess the quality of its source code according to what you have learned in the Software Engineering lecture (chapter 6). Fix some of its issues (at least three) and check-in your modifications. For example, you can rename variables or methods, break methods into smaller one, document methods or introduce enumeration types. Many IDEs, including Netbeans and Eclipse, provide tool support for refactoring Java programs. Your modifications must preserve the behavior of ImageJ. Report all related commits on the ticket before closing it.

2.5 Testing ImageJ

In Trac, there are 10 testing tasks available. Pick one of them (1 per group) and mark it as accepted (so that no assignment is solved twice). Prepare a test plan for your assignment and explain what kind of test it is and how you have chosen your test cases (you are free to choose your coverage criteria).

Automate your tests with JUnit 4[2]. For this purpose, create a new class in the `src/test/java` directory, so that your tests are run by Maven during the build. Before checking in your contribution, take note of the following points:

- If you need to use image files, you can place them in the `/src/test/resources` directory.
- You may have to modify ImageJ to perform your tests. If you do so, make your changes in such a way that it preserves the behavior of ImageJ from the viewpoint of an user.
- System tests can be implemented at the presentation or function level. Your tests can open windows and dialogs, but make sure that your tests close them programmatically, otherwise the continuous build may not terminate.
- Make sure that the project can be built correctly (tests can fail though, you are not supposed to repair the defects your tests have uncovered) before checking in your modifications. After the commit, verify that Hudson has built the project. If the build fails on the build server, it is very likely that it cannot be built by your colleagues either.
- Do not forget to include additional files in your commit (with the command `svn add`).
- In the commit's comment, refer to the Trac ticket. Once you have completed your task, close the ticket
- Report all related commits on the ticket.

Which difficulties have you encountered when automating your tests?

2.6 Wrapping Up

Once you have accomplished both the testing and improvement assignments, create a snapshot of the ImageJ project in the repository. The name of your tag must contain your Trac usernames.

Finally, have a look on the *timeline* view of Trac. Report all its entries related to your intervention on the source code of ImageJ.

[2]See `http://daiquiri.ifi.uzh.ch/trac/swq12/wiki/JUnit` for some documentation on JUnit 4.

3 Static Analysis

Static analysis of source code can help to hypothesize about the localization of a defect. In this exercise we explore the data and control dependencies in the `calculateXYZ(int n)` method (see Listing 1). Chapter 11, from the Software Engineering lecture, may reveal helpful for this task.

a) Draw the *program dependency graph* (PDG) of the method.
b) Based on the PDG, compute the following static slices:
   (a) forward slice for the `int y = 0` statement (line 2)
   (b) backward slice for the `System.out.println(x)` statement (line 18)
   (c) backward slice for the `System.out.println(z)` statement (line 20)
c) Is there any dead code (code that is never executed) in this method? Explain your answer by using the PDG. What other problems can be discovered with PDGs?

```
0  public static void calculateXYZ(int n){
1    int x = 0;
2    int y = 0;
3    int z = 1;
4    int i = n;
5    int j = n;
6    while (i>0){
7      x = x + 1;
8      i = i - 1;
9      j = i;
10     while (j>0){
11       if(j%2 == 0)
12         y = y + 1;
13       else
14         z = z * 2;
15       j = j- 1;
16     }
17   }
18   System.out.println(x);
19   System.out.println(y);
20   System.out.println(z);
21 }
```

Listing 1: Source code of `calculateXYZ(int n)`

4 Hypothesizing about a Defect

On the website of the exercises, you can find a Java program called `Inverse.jar.txt`. Download the file and rename it to `Inverse.jar`. This program inverses the digits of strictly positive integers. For example, 345 will be transformed into 543. You can launch this program with the following command:

```
java -jar Inverse.jar 345
```

It turns out that this program returns 103 as the inverse value of 300. Follow the scientific method (chapter 5, slide # 22) to come up with the best diagnostic (which inputs produce an error and what may be its cause) as possible. Report the complete history of your diagnostic in a table containing (a) your hypothesis, (b) your test cases and (c) the result of your test cases.