Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Software Quality and Dependability/ Analysis of Software Artifacts

DEI-FCTUC, 2020/2021

Assignment 2: Dynamic Software Testing

The goal of this assignment is to develop a software test plan and perform the testing campaign for a software code selected by the students (see details and requirements for selecting the code to be tested below in point 1). An important aspect is in the fact that the assignment should be regarded as an exploratory software testing exercise. That is, each group of students is expected to identify and select the most appropriate testing approaches, taking into account the topics covered in the T classes, and tools available at the Internet, instead of just following a predetermined testing script and using mandatory tools. Additionally, the focus is on ***dynamic testing***, which means that static approaches (e.g., static code analysis, code inspections, etc.) are out of the scope of the assignment.

The exploratory nature expected for this assignment also implies the necessary focus on the concepts, as a first step to identify and use the best testing solutions and tools. For example, it is useless to try to identify available tools to support control flow testing in a given program unit if you do not know what control flow testing is. For that reason, the (strong) recommendation is to start always from the concepts (types of tests, scope, etc.) and look at the assignment as an opportunity to learn how the concepts covered in the T classes can be applied in practice.

Although completeness in software testing is very important, the true goal of the assignment is to learn as much as possible. In other words, the evaluation of the assignment output will not be done in a "checklist" basis, taking into account only what has been actually done. On the contrary, all the evidence of lessons learned by the students will be taken into account and highly recognized in the evaluation of student's work. The discussion of the assignment in the defense will be particularly focused on the evaluation of what the students know about the work done in the assignment and not on the simple verification of the amount of work done.

1. Software to be tested

The code to be tested can be a unit that implements a given functionality in a larger software project or a small complete program. There are no restrictions about the programming language of the code, but the students should take into account that the Integrated Development Environments (IDE) available for the language and the features to support program testing in such IDEs, or the tools available at the Internet, such as tools to calculate complexity metrics, especially MaCabe metric, are dependent on the programming language. For some languages is much harder to find tools than for other languages. For example, if you have already used (at least in a partial way) unit testing frameworks such as JUnit, maybe is a good idea to select programs to be tested written in Java and try to explore JUnit as much as possible. The same for other testing frameworks and tools.

An important restriction in the selection of the software to be tested is that the code should have been produced by other people, and not by the students belonging to the group that is doing this assignment. Hybrid scenarios such as having one member of the group that has participated in the production in the code can be accepted, especially if the code developed by such member of the group can be included in unit testing (e.g., it is one or more functions or small modules).

Students can use code done by colleagues in assignments from other courses or can use code available on the Internet in code repositories. Obviously, the code should not be very large and should not be too small.

The focus should be on both **black box** and **white box** testing, and students are recommended to avoid selecting programs with large and complex interfaces (graphic interfaces), as the testing of interface intensive applications has specific aspects that are outside the scope of this assignment.

Since selecting the code to be tested is obviously very important, there is a first step that consists of submitting the source code that will be tested for approval. The proposed date for this first step is **April 24th** and the submission (in Inforestudante) should include the following:

- Names and emails of the members of the group
- A short text (half a page) describing the code that will be tested
- The source code

Important: since there are already many software testing assignments done by students in the past, an important criterion to approve the code proposed by each group is that it should not have been used in known testing projects before. **In other words, if you know that a given code has already been used in software testing assignments, do not propose such code, as it will not be accepted.**

2. Outcome of the assignment

The outcome of the assignment is the software test plan and the complete test suite that includes the test cases defined for the different testing strategies. Other artifacts such as specifications and code developed for testing purposes (or associated to specific testing tools or plugins) should be also submitted as part of the outcome of the assignment.

The suggested template (see below) for the software test plan is an adapted (and simplified) version of what is recommended by the ISO/IEC/IEEE 29119-3 international standard (part 3 – test documentation)[1] and the IEEE 829 standard for software and system test documentation[2]. In any case, it is worth mentioning that the suggested template should not

[1] See: http://www.softwaretestingstandard.org/part3.php

[2] See: http://img105.job1001.com/upload/adminnew/2015-02-04/1423058832-HCSCIRY.pdf

be regarded as mandatory, as the student groups are allowed to adjust the proposed structure of the software test plan if needed.

Test Plan Title

Test Plan Identifier

The goal is to identify uniquely the test plan in the organization/company. In general, companies have their own rules to generate these types of identifiers (ID). Very often, the test plan ID includes some structure to describe the level of the plan (concerning the software structure, master plan or plan for a specific part of the software) and the version of the test plan. This is important, as a test plan is a live document that evolves and normally has successive versions. It is also frequent to include the authors of the test plan and the contact information in the identifier section.

Example:
QCS-18/19-001-1.0   (course-school year-no group-version of the plan)
<names of the authors>
<emails of the authors>

1 – Introduction

Describes the purpose of the plan, identifies the target software, the level of the plan (master or sub-plan), the testing approaches, and provides the necessary context to read and understand the test plan. In practice, the introduction should provide a condensed view of the test plan (in an executive summary style, as it is not usual to include an independent executive summary section in the test plans).

2 – Software risk issues

Identify critical aspects of the software to be tested and potential risks. For example, extreme complexity, safety critical functions/applications, inclusion of third part products, poorly documented modules, unclear requirements, etc. This section should briefly evaluate the potential impact of these issues in the testing activities and try to identify mitigation actions in the testing strategy. Additionally, the possible impact of imperfect testing (and the consequent bugs that may remain in the software after deployment) should be evaluated as part of the risk.[3]

3 – Items and features to be tested

Presents the list of things to be tested in the scope of the test plan. It is important to identify clearly the artifacts/elements to be tested, including versions and configuration requirements, if any. This section should take into account the different levels considered in the test plan in order to identify the units or modules to be tested. In addition to this technical view describing the items to be tested, this section should also present the features to be tested from the end user point of view. This is concise description of what the system does, from the end user perspective. Although the functional specification can be used as input for the description of what the system does, the idea here is to provide a concise and high level description of the functionalities, as seen by the end user.

4 – Items and features not to be tested

[3] Remember that the classic equation for the assessment of the risk includes two terms: the probability of an undesirable event multiplied by the adjudged impact of such event.

Describes the items of the system and the features that are not covered by the testing plan, including both the system view and the end user (functional) view. When some elements or features are not tested it is usual to provide a short explanation why such items/features are excluded (e.g., items that have been used before and are considered stable, thus representing a low risk). It is worth mentioning that students should provide a good description of the features of the software under test. If there is a requirement document available with a feature list, then this chapter can point to the specific features of those documents.

5 – Testing Approach

Presents the overall testing strategy for the test plan, including the testing techniques used for the different levels of testing, coverage requirements, testing setup, possible software configurations to be tested, tools used in the testing process, metrics to be collected, etc.

6 – Item Pass/Fail Criteria

Defines the completion criteria for the test plan. This is a very important aspect of the test plan and should consider the different testing levels. For example, at unit testing level the completion criteria could include the execution of all test cases, a given code coverage achieved, or a target for the number of defects exposed (e.g., zero major defects and less than a given target number of minor defects). At system level testing, the completion criterion includes typical the execution of all test cases and the achievement of a given target goal concerning the number and severity of defects exposed by the testing. The criteria to decide if a defect is major or minor must be defined as well. Normally, the severity is related to the possibility of the defect to originate a failure.

7 – Test Deliverables

Identifies what is to be delivered as part (and consequence) of the test plan. This includes the test plan itself, the test cases, test design specifications, code developed for testing purposes, configurations and settings of specific tools used, the outputs produced by the tools used in the testing, error logs, and the test completion report including the list of problems (defects) identified.

8 – Environmental Needs

Identifies specific needs (if any) for the execution of the test, including hardware, software, tools, etc.

9 – Staffing and responsibilities

Describes the human resources needs for the testing and identifies the different roles and responsibilities. Other aspects related to the testing team, such as specific training needs, if any, should also be described in this section.

8 – Test Completion Report

Reports the results of the tests at the different levels, describing the defects identified and the test result according to the pass/fail criteria defined in section 6. It is common to use tables to provide a condensed view on the test results.

<end of suggested template>

3. Resources

Students are supposed to use their own computers and virtual machines in the systems infrastructure of the Department. As already mentioned, students are encouraged to search for testing tools on the Internet. Tools such as JUnit for Java programs are obvious choices, as well as plug-ins for Eclipse for code coverage visualization and control flow graph generation[4]. This is the most exploratory part of the assignment, as the students are expected to search, select, evaluate and use available testing tools. Both open-source tools and commercial tools available on a trial basis can be considered. Although Google is probably the best place to start this quest, specific sites that include lists of hundreds of testing tools are also a good starting point (e.g., http://www.softwareqatest.com/ includes 500+ tools – but most of them are not relevant for the purposes of this assignment).

An important aspect when searching and evaluating possible testing tools to be used in the assignment is the (already mentioned) focus on the concepts. For example, knowing that it is recommended in the assignment to use boundary value analysis and equivalence partitioning for the black box testing and control flow testing (and maybe data flow testing) for the white box testing, the first thing the students should do is to learn the concepts about these testing techniques (start by the slides used in the T classes). Once the students know the testing techniques, the search and identification of adequate tools is much easier.

4. Calendar and miscellaneous

This assignment must be done by the same groups already formed for the assignment 1. Changes in the composition of the groups must be reported.

The outline of the calendar is the following:

- Week of April 19th – Presentation of the assignment, questions from the students and discussion (PL classes)
- April 24th – Deadline to submit the code to be tested (Inforestudante)
- PL classes for the rest of semester – Provide support to the students and follow the progress of each group (the assignment cannot be done in the final days before the deadline!). Please, see the plan of the classes available at Inforestudante.
- May 28th – Deadline of the assignment (Inforestudante). Note that this date is a week later than the deadline defined in the master plan in the beginning of the semester.
- Week of May 31st – Assignment defenses. Students' groups will select the day and time for their defense among a large number of choices.

[4] http://www.methodsandtools.com/tools/pesttunittesting.php
http://coverlipse.sourceforge.net/index.php
http://eclipsefcg.sourceforge.net/index.html