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

Experimental Methods in Computer Science
(and in Informatics Engineering)

DEI-FCTUC, 2016/2017

Assignment 2 – Design of experiments

This assignment is an exercise of designing an experiment. Although the main goal is pedagogic, which naturally force us to simplify several aspect of the proposed experiment, the assignment includes all the steps of a real computer systems/software experiment.

There are many types of experiments and the term "design of experiments" (often referred as experimental design as well) is used for the process of defining and planning experiments in such a way that the data obtained can lead (after some analysis that depend on the nature of the experiment) to technically and scientifically valid conclusions.

The topic of designing experiments of different types and for different purposes has been addressed in detail in the lectures (T classes). In any case, the next paragraphs present a brief overview before describing the assignment proposed.

A classic experiment includes the following elements (you can take them as basic steps for the purpose of this assignment):

1. Problem statement (or research question)
2. Identify variables
3. Generate hypothesis
4. Define the experimental setup/scenario
5. Develop the tools and procedures required to run the experiment
6. Run the experiments, collect the data/measurements and perform basic statistical treatment to the measurements
7. Perform data analysis
8. Draw conclusions (and often go back to the beginning and reformulate the problem statement or test a different hypothesis)

The formulation of sound problem statements (or research questions) is not easy and this is particularly true for the computer science and informatics engineering field. A good (i.e., relevant problem statement) should be focused enough to allow the clear

identification of the variables of the problem but, at the same time, should be sufficiently open to allow different hypothesis to answer the problem/question.

An example of formulation of a possible problem statement is:

How does the setting (noise, temperature, music, open space, etc.) of the room used by programmers affect the number of bugs found by test suits in the modules produced by such programmers?

The effect we want to measure (dependent variable) is the number of bugs found and the factors (independent variables) are the different room settings and situations. It is assumed that there is a set of conditions that remain stable (e.g., type of programing language, complexity of the modules, etc.). The basic type of experiments only changes one factor (i.e., independent variable) at the time. More complex experiments use a factorial approach in which several factors are changed together to make the experiment more efficient and allow the study of possible interactions among factors. For the time being, in the context of the present assignment, the recommendation is to consider only one factor at the time.

A possible hypothesis for the problem statement mentioned above is that programmers tend to make fewer bugs if they listen classic music at a confortable volume while programming. In this case, the hypothesis is directional (i.e., establish a direction for the dependency between the dependent variable and the independent variable) but the hypothesis could simply state that music influences the number of bugs (non-directional hypothesis).

Even in a simple example like this, it is easy to understand the difficulties associated to the correct validation of hypothesis. For example, it is necessary to perform experiments with a representative group of programmers, use a variety of software under development, assure that the conditions of the different experiment runs remain stable, etc. The analysis of the results in order to check if their distribution fit the hypothesis requires adequate statistic testing techniques, as has been presented and discussed in the lectures (T classes). In most of the cases, the assumption that the observations (i.e., results directly gathered from the experiment) follow approximately the normal distribution (or a T student distribution) greatly simplifies the test statistics. But care is necessary to certify that this assumption is really valid for the experiment at hand.

1. Goals, assumptions and recommendations

The purpose of the assignment is to design an experiment and execute all the steps to evaluate the following problem statement:

What is the probability of a test suite to detect the software faults (i.e., the bugs) that still exist in the code of a program?

The goal of the experiment is to assess the quality of test suits in terms of the probability (coverage) of the test suit (or test suits) under evaluation to detect possible bugs in the program under testing. It is expected that a test suit can detect all the residual bugs in a given program but, as we know, this is not often the case.

Thus, the objective of the assignment is to design, build and run an experiment that allows the following:

1. Assess the probability of bug detection of typical test suits developed for real programs (see below the type of programs to be considered).
2. Formulate a hypothesis (e.g., assuming a given target for the bug detection probability) and test the hypothesis, in order to discuss the consequences of the result of such hypothesis testing.

As happen most of the cases in experiment design, it is necessary to learn/know the concepts about the topic or subject of the experiment. In the present assignment, the topic selected is software testing. Thus, it is time for the students to refresh their knowledge on software testing or learn some new concepts on this topic. The next paragraphs present some basic information on testing, as well as some specific suggestions for the proposed experiment. The students should research for additional information on software testing (or ask the teacher for help… but only when students feel that they are stuck on a problem and cannot find a solution).

We will consider a typical testing scenario in which the developer submits the program (or program module) he/she just developed to be tested by the testing team. In industrial software development, tests are normally carried out by a separated team (testing team), not only because testing is a highly specialized job but also to assure some independence between the developer and the tester.

A **test suite** is a collection of **test cases**. The test case is the basic unit in informal software testing (there are also formal testing methods but those are not considered for this assignment). A **test cases consists of two basic parts**:

- Set of inputs to be submitted to the program under test.
- Expected program outcome (results or program behavior).

The mechanism that uses the expected outcome of the test cases to determine whether the program under test has passed or failed such test case is known as a **test oracle**.

The goal of defining a good test suite is to find a set of test cases that assures, as much as possible, that possible bugs in the program under test are detected by one or more test cases. The number of test cases needed for a good test suite can vary a lot, ranging from just a few tens to hundreds or thousands of test cases. There are many strategies and tools to define test cases and, if needed, students can easily find tutorials and documents on software testing in the Internet (but resist to the temptation of just looking at the Wikipedia).

There are two main types of testing methods that differ in the way the test cases are defined:

- **Black box testing**, when test cases are defined considering only the functional specification (what the program does without looking at the code of the program under test). Test cases are informally defined taking into account the input parameters and their domains. In general, the test cases cover typical values for each input parameter, values in the boundaries (of the domain) and values outside the domain of the parameters to test the robustness of the program as well.

- **White box testing**, when the test cases are defined taking advantage of the knowledge of the program code, in order to maximize the coverage in terms of code execution (i.e., to assure that the entire code and data are adequately tested). The two main types of white box testing are **control flow testing** (test cases are defined to assure that all the program paths are tested) and **data flow testing** (test cases are defined to assure that all program variables are tested).

The following assumptions and recommendations should be taken into account in the work:

- The students must select the programs to be used in the experiment. Any existing program can potentially be used, provided that it fits into the following criteria:
    - Not very small (at least 150 lines of source code) but, ideally, not very large and complex.
    - The programs should use a command line to input the parameters or any other simple way to input the parameters (in other words, avoid programs with heavy interactive interface) and should produce a deterministic output (to keep the test oracle simple).
    - Ideally, the programs should be written in C. If the students choose programs written in another language it will be required an addition step (see below the software fault injection bullet).
- The students should define a test suite for each program used in the experiments (ideally, the experiments should be performed with more than one program). The recommendation is to use a simple (and quite intuitive) black box testing approach to define the test cases of the test suite(s). If the programs have test suites available (some programs available in free repositories in the Internet include test suites for regression testing), then the students can use such test suites (i.e., do not need to define the test suites).
- The key idea to perform the experiment and verify whether the test suit can detect bugs in the program under test is to inject bugs in the source code (to simulate a real bug that could be there) and then run the test suite to verify if at least one test case can detect the injected bug. This technique is called software fault injection (or defect seeding when used in code inspections). This is similar to a technique called software mutation, but software fault injection simulate realistic bugs while mutation just uses any change (mutation) in the program that is syntactically acceptable.
- The basic step of the experiment (to collect a sample measurement) consist of injecting a software fault (bug) in the program and then run the test suite to determine if the bug was detected by at least one test case of the test suite. And repeat this basic step for a given number of faults in each program, in order to estimate the probability of the test suit to detect bugs in the program under test.
- The faults should be injecting using a published list of Top N (N=13) of most common software fault types for programs in C, resulting from field studies of real bugs. The list is included as an Annex to this document. If the programs selected by the students are not written in C, the student will have an extra step to adapt the list of the Top N of most common software faults to the language used in the selected programs.

- Different groups of students can use the same program(s) and can share test suites as well. But the set of faults used to simulate the bugs must be done by each group and must not be shared among the student groups.
- In order to save time and effort, students should developed scripts or simple tools to automatize as much as possible the different tasks necessary for the experiment setup.

2. Outcome

The outcome of the assignment is a written report (PDF file). The report should describe all the steps of the design of the experiment with enough detail to allow others to reproduce the experiment and, of course, should provide clear conclusions about the problem statement.

Given that the sole outcome of the assignment is a report, the quality of the document (i.e., structure of the report, precision of the results reported, quality of writing) is of paramount importance. There is no suggested template for the report structure. The goal is to avoid the rather passive situation in which the students just fill in a given report structure. On the contrary, defining the best structure for the report is part of the goals of the assignment. The teacher is fully available to discuss the proposed structure with the students, as well as any other aspect of the report.

In addition to the report, the students must keep a folder with all the programs, scripts, and tools used or developed for this assignment. This folder could be useful during the final assignment defense. Furthermore, the experimental setup should be ready to run during assignment defense, in case it is necessary to show any specific element of the experiment.

3. Resources

Students are supposed to use their own computers or virtual machines provided by the Department. Free software repositories available in the Internet are the most obvious source of programs to be used in this experiment. But any other source of existing programs is acceptable.

The software faults can be injected manually (one fault at the time) in the programs under test using the information provided in the Annex of this assignment, which describes the fault operators to the Top N most common software fault types.

In alternative to the manual fault injection, students can use the software fault injection tool developed by Gonçalo Pereira (2016) in his Master thesis. The instructions and examples on how to use the tool are available in the following link:

https://ucxception.dei.uc.pt/index.php/software-faults/

An account is available at ucx.dei.uc.pt with the following credentials:

USER: guest

PASS: ucXception$user

4. Calendar and miscellaneous

The students groups already defined for Assignment #1 must be maintained for this assignment. The work is expected to progress according to the following calendar that will be verified in the PL classes:

- **Week of November 14tth**: All the variables of the experiment should be clearly identified and all the levels decided for the independent variable should also be identified. The set of programs to be used should be defined. The test suites for the programs should be under development (if not finished) in this week.
- **Week of November 21th**: The experimental setup should finished and ready to start the experiments.
- **Week of November 28th**: The students should have already some experiment results. Some analysis is also expected to confirm if the experimental setup is satisfactory or if it is necessary to change or adjust any element of the experiment.
- **Week of December 5th**: Most of experiment results should be available by this week. The students should be work on the hypothesis testing.
- **Week of December 12th**: The students should be working on the report by this week.
- The assignment must be submitted (PDF file submitted at Inforestudante) before **December 22nd at 23:59**.
- Oral defense (all assignments) will be scheduled to the first two weeks of January. The actual schedule will be made available in early December.

Plagiarism means mandatory fail in the course and internal (UC) disciplinary procedure. Please, refer adequately all text and material you take from the Internet. All parts of the report must be written by the students and not copied & pasted & changed from the Internet.