Prof. Dr. A. Podelski
Dr. V. Langenfeld
E. Henkel

Summer term 2024

Softwaretechnik/Software Engineering

`https://swt.informatik.uni-freiburg.de/teaching/SS2024/swtvl`

Exercise Sheet 1

Submission: Wednesday, 2024-04-24, 15:59

Exercise 1 – Does Software Development Success Matter? (10/15)

Consider the cases A–C presented in the appendix. Each case is a well-known example of an incident where using software lead to a considerable damage. The development of those softwares can hence be argued to be unsuccessful.

1.1) Research a different, recent (that is, at most five year back from now) incident *without loss of life* and report the incident following the style of the cases in the appendix:

- Provide a general description of the case, followed by a more detailed description of the software-related issue and its consequences.
- Quantify the damages caused and argue why the case is relevant.
- Provide the source(s) of your information. (3)

1.2) For your own example from Task (1.1), argue in how far the software development would be considered successful or unsuccessful (in the sense of the lecture) by the three main kinds of stakeholders. (3)

1.3) For each of the cases A–C and your own example from Task (1.1), discuss in how far the incident (following official reports, or in your opinion) is related to issues with requirements, design, quality assurance, or management. (4)

Exercise 2 – Lines of Code Metrics (5/15)

Consider the following lines of code (LOC) metrics:

- LOC_{tot} = Total number of lines of code.
- LOC_{ne} = Number of non-empty lines of code.
- LOC_{pars} = Number of lines of code that do not consist entirely of comments or non-printable characters.

2.1) Calculate the value of the LOC metrics for the Java program in the file `MyQuickSort.java` that accompanies this exercise sheet. (2)

2.2) The LOC metrics are often used as derived measure for the complexity or effort required to develop the code being measured.

In particular the family of LOC metrics is notorious for being subvertible. If a metric is subvertible, its value can be manipulated to increase or decrease it arbitrarily while preserving the same program semantics. I.e., for every program, there always exists a semantically equivalent program (that performs the same computation, and thus should have needed roughly similar effort to develop) that has substantially different metric values.

Convince yourself of this claim for the case of LOC_{pars}:

(a) Give two semantically equivalent programs (in a high-level programming language of your choice, like Java, C++, C) with substantially (at least an order of magnitude) different metric values. (1)

(b) Is your example a rare exception? If not, give a procedure to subvert given programs to a given metric value; if yes, argue why. (1)

2.3) What is the largest value of the[1] LOC metric for any of your contributions to a software project so far? What are the values of the LOC metric for the whole project? (An assignment in a programming course also counts as a project.) (1)

Give just the order of magnitude of the two figures per team member according to the following intervals:

	(a)		(b)		(c)		(d)		(e)		(f)		(g)		(h)		(i)	
0		30		100		300		10^3		$3 \cdot 10^3$		10^4		10^5		10^6		∞

For example, if a team member's situation were "ca. 2,500 lines in a ca. 500,000 lines framework", then he or she would report (e) : $[10^3, 3 \cdot 10^3)$ *for contribution and (h)* : $[10^5, 10^6)$ *for whole project (as part of complete sentences, of course).*

[1] Choose the one of the three that you consider most appropriate for this task.

Appendix

Consider the following cases where software failures caused important damage.

A. Therac-25 (1985-87)

The radiation therapy machine Therac-25 erroneously delivered extremely high radiation doses in at least six incidents between 1985 and 1987, causing serious injuries to the patients being treated.
The dangerous radiation overdoses occurred when the machine activated a high energy electron beam without appropriately having rotated four hardware components onto the path of the beam that would condition it to make it safe for patient treatment, thus allowing the electron beam to directly hit the patient and deliver a potentially lethal radiation dose, 100 times larger than the intended dose.
The error was caused by a race condition between the data entry routine, which communicated with a keyboard and a terminal screen to configure the machine, and the routine that monitors radiation treatment. If the operator changed the configuration using the terminal in less than 8 seconds while the machine was rotating a magnet plate into place, the changes entered would go undetected and a flag indicating that the configuration is complete was set, allowing the machine to continue the therapy sequence with the wrong parameters. After the initial incidents, the software was modified to fix the specific race condition; other problems of concurrent programming due to the poor design of the software allowed similar incidents to still occur.
The machine caused serious injuries to six patients, three of which later died from complications of the radiation burns. The damage caused by the software error has both moral and legal implications: human life was lost as a result of poor software development practices. Furthermore, the company who developed the machine was faced with several lawsuits and subsequent economic losses derived from settling the suits in and out of court.
Source: `https://www.cs.jhu.edu/~cis/cista/445/Lectures/Therac.pdf`

B. Ariane-5 Rocket (1996)

On June 4, 1996, the first flight of an Ariane-5 rocket from the European Space Agency (ESA) was scheduled to launch a group of satellites (the *Cluster* spacecraft) into orbit to study the earth's magnetosphere. Thirty-seven seconds after liftoff, the rocket veered off its programmed flight path, broke up, and exploded.
The incident was caused by the adoption of a software module for the inertial control system from the rocket's previous generation. The newer rocket was subject to a greater horizontal acceleration, which caused an overflow when converting a 64-bit floating point value into a 16-bit signed integer value, thus triggering a hardware exception, which in turn caused the inertial reference system to enter a failure state and stop providing valid attitude information.
The approximate damage caused by this error amounts to approximately 1 billion US$, contributed by the taxpayers of the states participating in ESA, plus the delay and cost increase of the scientific mission on board.
Source: `http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf`

C. Toll Collect (2003-06)

The German highway toll collection system for heavy traffic "Toll Collect" was scheduled to enter service in August of 2003. The consortium in charge of developing the system failed to meet the deadline. The system only entered its initial reduced load operation in January 2005, delayed almost a year and a half. The system operates in full functionality since January 2006.
The project planning, which involved around twelve different development partners at twelve different locations, was faulty. No standards for the interaction between the software modules were set, ranging from the access to databases to the graphical user interfaces. This situation made integration tests very difficult, because the developed software modules were, in some cases, incompatible.
The damage caused by the faulty project extends from the developing firms, who incurred contract penalties in the hundreds of millions of euros, to the taxpayers and the state, who provided the resources for the development, and missed an undetermined amount of revenue in toll fees that may be well into the billions of euros.
Source: `http://www.sueddeutsche.de/wirtschaft/chronik-der-autobahn-maut-pleiten-pech-und-pannen-1.508682`