Aufgabe 11

Nathan Ritter
5566519
Lorenzo Tecchia
5581906
2023.06.27

Contents

Chapter 1

Task 1

(a) I, personally don't do: 1,5,9, 11, 12, 14, 15, 19, 22 ,24 , 25.

(b)

(c) Number 30, if one's system has a large capability of RAM and not having flushed the "data" loaded into the RAM doesn't not affect the performance of the machine, then it could go unnoticed but still would have occupied a big chunk of a system memory and the running of a test just one time wouldn't notice the abundance of memory allocation. Also number 29 since not closing a DB connection for example would not affect the execution of the machine at first but a second query to said database would result into an error if before the start of the query one would try to open a new connection to said database or any other external source.

(d) These are the three identified:

1. The "Need to know" principle (abstraction, encapsulations of information) wrongful access to sensitive data can incur into faults, so there is a need for instruments of encapsulation like strong typing and in general creation of classes. [3](Paragraph 20.2.1)

2. The lack of error prone language constructs. Some language have deprecated or choose to not implement at all some of these error prone structures like Java doesn't implement pointers and uses the garbage collector in response to dynamic memory allocation, but still has some prominent errore prone features like the handling of floating point numbers. [3]

3. Is the system using the fixed-point numbers instead of floating point ones? As a response to the last mentioned problem, fixed-point numbers are represented to a given number of a decimal place. [3]

(e) Citing (Lutz1993)[2] and (Fagan1976)[1]:

> "difficulties with requirements are the key root cause of the safety-related software errors, which have persisted until integration and system testing".

In embedded systems, is reported that more than 90% of defects can be discovered in program inspections. So when requirements are critical and to be accounted general reviews are much more effective and useful than automated system.

Software quality is not just about whether the software functionality has been correctly implemented, but also depends on non-functional system attributes that automated testing cannot verify.

Automated testing solely rely on how the test inside the code is implemented and can exclusively be carried out with the code being executed, whereas static review are done by a team often very experienced of developers which can spot major defects inside the code and not depend on any pre-define test. If there were only automated and dynamic tests the quality of the software would be just as good as the quality of the tests.

Chapter 2

Task 2

Chapter 3

Task 3

Bibliography

[1]   Michael Fagan. "Design and Code Inspections to Reduce Errors in Program Development". In: *Software Pioneers: Contributions to Software Engineering*. Ed. by Manfred Broy and Ernst Denert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 575–607. ISBN: 978-3-642-59412-0. DOI: `10.1007/978-3-642-59412-0_35`. URL: `https://doi.org/10.1007/978-3-642-59412-0_35`.

[2]   R.R. Lutz. "Analyzing software requirements errors in safety-critical, embedded systems". In: *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*. 1993, pp. 126–133. DOI: `10.1109/ISRE.1993.324825`.

[3]   Ian Sommerville. *Software engineering*. 7th ed. Boston: Pearson/Addison-Wesley, 2004. ISBN: 0321210263.