ECE 322

Assignment 1

Arun Woosaree
September 22, 2019

Arun Woosaree XXXXXXX

1

After reading the two papers, the two most essential factors which make software testing difficult in my opinion is:

1. Human nature
   In my opinion, people are generally lazy. In terms of software testing, this means that in some cases, people will tend to write enough tests to convince themselves that their program is correct, or In some cases they might even refuse to write tests once they have some code that in their mind already works, so there is no perceivedneed to test the software. This is probably because most programmers experience a significant mental reward when they finally get a program 'working'. However, when it comes to writing tests for the code they just wrote, there usually is much less excitement. Sometimes, programmers may even find themselves wrestling with a testing framework, which can be a frustrating experience when (they think that) their program is already working as expected. This is a non-technical factor.

2. The always changing nature of software
   It is nearly, if not impossible to write software once, ship it to a customer, and forget about it. For example, new hardware might be released, and users may want the software to interact with new hardware. Maybe someone else releases new software, and your users demand Interoperability with it. There can be other factors too, such as new laws and regulations that the software must comply with, even though the software was released before the regulations were realized. Not only does this make software difficult to maintain, since factors like these cause the requirements to change, but because the requirements are changing, the tests have to be updated or extended as well. Assuming the software engineers wrote their codebase in a maintainable fashion, updating the software to conform with society's new expectations *usually* is not that difficult. However, one small change in how the software behaves may result in a plethora of tests failing since they were written at a time when the requirements were different. When this happens, software engineers may become desensitized to the failing tests, or even disable them and allow 'tech debt' to pile up, in addition to the need to write new tests. These factors combined can make software testing difficult in nature. This is a technical factor.

2

- Functionality
  - Does the software allow the vehicle to work autonomously, without human input?
  - Does it do so in an acceptable manner?

  - (i.e. does it reach the destination in about the same amount of time as a good human driver would do or better?
    Does it do it as safe as, or better than a good human driver?)
  - Can a human driver override the autonomous controls if needed?

- Performance and reliability
  - How reliably does the software make the vehicle react to its environment?
  - Does the software still control the vehicle in an acceptable manner in more difficult situations?
  - If certain conditions like heavier traffic puts a higher load on the processing unit for the system, does the software still behave reliably?
  - It should work in different driving conditions (e.g. highways vs in-city, sunny vs slippery roads)
  - Will the software perform just as reliably a few years from now? (or better due to software updates)

- Efficiency
  - Does the software utilize its efficiently? (i.e. How much CPU, GPU, memory etc. *should* it need versus how much the software is actually using)
  - Does the software allow the vehicle to react quickly enough to its environment?
  - Are the algorithms used in the software the most efficient? (e.g. could do big O time-space analysis)

- Maintainability
  - Is the software well-documented?
  - How easy is it to add functionality to the software? (e.g. if new driving laws have to be followed, how easy will it be to add a patch to be in compliance)
  - How much technical debt exists in the software project
  - When faults are found, how easy is it to fix them?
  - How are updates shipped to the consumer? Can they be done over-the-air (OTA), or does the car need to routinely visit a car dealer/autobody location to get updates?

- Usability
  - From the user's perspective, how easy is it to use the autonomous mode of the vehicle, and how is the user experience?

  - Also, how easy it is to switch between autonomous and manual modes of the vehicle?

- Portability
  - Can the software be used in multiple types of vehicles?
  - (e.g. if there are multiple models of cars, can the same software be used with all of them? What about trucks or vans?)
  - How easy would it be to port the software to a newer version of a car, so that everything need not be written from scratch when the car manufacturer wants to make a new model?
  - Hardware compatibility: e.g. does the software work on multiple types of architectures? (e.g. x86, ARM, RISC-V, etc.)

Risk Category	Technical Risk	Business Risk
Functionality		
Car might crash while in autonomous mode	(3)	(1)
What if the user cannot manually override the autonomous controls?	(5)	1
Performance & Reliability		
Software not as reliable in different situations (traffic, weather...)	(2)	(1)
The vehicle does not perform as reliably in a few years as it did when it was new	(3)	(2)
Efficiency		
Software does not utilize its hardware resources efficiently	(2)	(5)
Maintainability		
Poor software documentation	(1)	(5)
Technical debt	(1)	(5)
Poorly architectured code base (i.e. not easily extendable)	(1)	(4)
Ease of delivering fixes and software updates	(3)	(2)
Usability		
Software is not intuitive for the user	(4)	(2)
Portability		
Software cannot be ported to newer hardware or other vehicles	(1)	(3)
Other		
Poor test coverage	(3)	(5)
Poor quality tests	(1)	(4)
Poorly defined requirements	(2)	(3)
Sensor or other hardware malfunction (also, how the software handles faulty hardware)	(2)	(1)
Security and hacking	(2)	(1)
Compliance with the law	(5)	(1)
Privacy risks (collecting camera data, customer data potentially leaking)	(4)	(2)

3

1. **Reliability** (Product Operation factor)
   chosen because of the requirement with the probability of a state of failure

2. **Interoperability** (Product Transition factor)

chosen because of the requirement to interface with other software

3. **Usability** (Product Operation factor)
   chosen because of training requirements (< 3 days), and operation usability (being able to manage 20 patients per hour)

4. **Efficiency** (Product Operation factor)
   chosen because of the hardware resources required (serving 12 workstations, 8 testing machines. . .)

4

Feature Description

Boeing 737 MAX flight control system

To make their new plane more efficient, Boeing added larger engines to the 737 MAX. To compensate for ground clearance, they moved the engines further in front of the wings, which makes it so that the engines can provide lift at higher angles of attack, in addition to the wings. This makes changing the plane's pitch more difficult. Boeing's "solution" to this problem was in their software, so-called "Maneuvering Characteristics Augmentation System"[1] The software's purpose is to make the plane tilt down if it "thinks" that the plane is going to stall. Unfortunately, when the software takes this corrective action, it does not give pilots an easy way to override it if they look out the window and conclude that the plane is in fact, not stalling. Apparently, it was "only" classified as a "major failure", meaning that it could cause physical distress to people on the plane, but not death.[2]

Nature of Software Failure

1. The *Maneuvering Characteristics Augmentation System* had no redundancy. Normally, systems like this have 2 redundant computers collecting data from different sensors, and if there is a disagreement, the software does a "sanity" check, and also alerts the user. In this case, if the software thought something was wrong, it would immediately act on behalf of the pilot, which should have been fine, but it was difficult for the pilots to maintain control as the failure would happen frequently and the system was physically challenging to override.[3]

2. The sensors used to determine if the plane is stalling are known to be less reliable than other sensors on the plane. Normally, this isn't a problem, as the pilots and software are supposed to be able to do "sanity" checks on the reported sensor data, but it was a problem in this case since the system made it difficult for the pilot to override the controls.[1]

3. The plane itself is packed with a whole bunch of modern features, yet most of it is designed to handle and work like the original 737 which is roughly

50 years old. This was done so that pilots did not have to get re-certified to use a new plane design. Originally, it did not handle like the other 737 models, and the software was added to attempt to make up for this difference.[3]

Any testing efforts regarding the failure?

Virtual simulations were done, and the issue of the plane repeatedly nosediving was noticed in 2017, before the plane was certified. However, Boeing concluded that the problem did not "adversely impact airplane safety or operation.", and that "pilots could overcome the nose-down movement by performing a procedure to shut off the motor driving the stabilizer movement."[4][5]

Any follow up action taken? Any plan to alleviate further problems?

Boeing should be rolling out a set of software updates[1][4]. There is no confirmation, only speculation that the software update will cross-check data between the two redundant computer systems instead of immediately nosediving.[1] Pilot training will also be undergoing an update.[6]

References

[1] *How the boeing 737 max disaster looks to a software developer*, Apr. 2019. [Online]. Available: https://spectrum.ieee.org/aerospace/aviation/how-the-boeing-737-max-disaster-looks-to-a-software-developer.

[2] D. Gates, *Flawed analysis, failed oversight: How Boeing, FAA certified the suspect 737 MAX flight control system*, Mar. 2019. [Online]. Available: https://www.seattletimes.com/business/boeing-aerospace/failed-certification-faa-missed-safety-issues-in-the-737-max-system-implicated-in-the-lion-air-crash/.

[3] D. Campbell, *The many human errors that brought down the boeing 737 max*, May 2019. [Online]. Available: https://www.theverge.com/2019/5/2/18518176/boeing-737-max-crash-problems-human-error-mcas-faa.

[4] *Latest 737 max fault that alarmed test pilots rooted in software*, Jul. 2019. [Online]. Available: https://www.bloomberg.com/news/articles/2019-07-27/latest-737-max-fault-that-alarmed-test-pilots-rooted-in-software.

[5] B. Goggin, *Boeing reportedly knew of the software error on the 737 max for a year before telling airlines and regulators*, May 2019. [Online]. Available: https://www.businessinsider.com/boeing-knew-737-max-software-error-year-before-telling-faa-2019-5.

[6] S. Prokupecz, D. Griffin, and G. Wallace, *New flaw discovered on boeing 737 max, sources say*, Jun. 2019. [Online]. Available: `https://www.cnn.com/2019/06/26/politics/boeing-737-max-flaw/index.html`.