Ioana-Gabriela Chelaru
248/1

QA Project

Project used:

My bachelor's thesis application, which is a small cross-platform application written in python, can observe a subject's facial features from a continuous video stream and detect its early signs of drowsiness. The targeted users are mainly drivers, the end goal is eliminating the risk of falling asleep while driving, therefore reducing the number of car crashes.

Software quality model selected:

ISO 9126

Evolution of SQ Models

Functionality	Reliability	Usability	Efficiency	Maintainab.	Portability
Suitability	Maturity	Understand	Time behav.	Analysability	Adaptability
Accurateness	Fault tolerance	Learnability	Resource beh.	Changebility	Instability
Interoperability	Recoverability	Operability		Stability	Conformance
Compliance				Testability	Replacebility
Security					

I chose the model **ISO 9126**, because it seemed the most simple, yet fitting for my project since it contained all the factors that I was interested in. I chose this one, rather than *ISO 25010*, because I wasn't interested in the other two aspects introduced by the second one, more specifically *security* and *compatibility*.

Software factors to be evaluated:

For my application, I have found two very important factors: **functionality** and **portability.** These two are probably the most important aspects of my application since its success is based on its accuracy and meeting the requirements' standards and its capacity to be easily transferred and used on different devices and different environments.

The third factor that I'll be looking into is **reliability**. I am interested in seeing some scores on how durable my application is, considering that I have designed it with the idea in mind that it would be capable of running smoothly for hours, let's say at least a drive of 15 hours.

Functionality

It allows drawing conclusions about how well software provides desired functions. It can be used for assessing, controlling, and predicting the extent to which the software product (or parts of it) in question satisfies functional requirements.

- **suitability:** how suitable software is for a particular purpose; it correlates with metrics that measure attributes of software that allow concluding the presence and appropriateness of a set of functions for specified tasks.
- **accuracy:** how well software achieves correct or agreeable results; it correlates with metrics which measure attributes of software that allow concluding about its provision of correct or agreeable results.
- **interoperability:** how well software interacts with designated systems; it correlates with metrics which measure attributes of software that allow concluding about its ability to interact with specified systems.
- **security:** how secure software is; it correlates with metrics that measure attributes of software that allow concluding about its ability to prevent unauthorized access, whether accidental or deliberate, to programs or data.

Bellow, I simulated a requirements checklist for my application, in order to analyze how many of the requirements that I had in mind when designing the application, actually made it in the final product. It can be seen that the basic requirements are met, but the system struggles with users that wear glasses that have a thicker frame because it will cause confusion, but perhaps a bigger issue is that the application is bias, in the sense that it will perform poorly on asian or black users since the data that was used for training did not include such cases.

	Requirements	**Fulfilled**
1.	The system should be able to receive as input a continuous video stream, process it and output a prediction	✓
2.	The system should support both users with glasses and without glasses	X (only transparent lenses)
3.	The system should differentiate blinking from signs of drowsiness	✓
4.	The system should alert the user when signs of fatigue are detected	✓
5.	The system should give the same result for all users in the same circumstances	X

Next, I took a look at the analysis done in Lab 1 and 2, more specifically the metrics that Multimeter provides and are related to the functionality of the system, which are:

- lines of code
- McCabe cyclomatic complexity
- number of attributes and methods
- number of local methods
- change dependency between classes
- change dependency of classes, etc.

"comment_ratio": 24.781968512855364,
"cyclomatic_complexity": 14,
"fanout_external": 11,
"fanout_internal": 1,
"halstead_bugprop": 3.5723852727715704,
"halstead_difficulty": 48.729559748427675,
"halstead_effort": 522242.284781776,
"halstead_timerequired": 29013.460265654223,
"halstead_volume": 10717.155818314712,
"lang": [
"Python"
],
"loc": 304,
"maintainability_index": 26.910225600592355,
"operands_sum": 596,
"operands_uniq": 159,
"operators_sum": 827,
"operators_uniq": 26,
"pylint": 100.0,
"tiobe": 84.86937283570603,
"tiobe_compiler": 100.0,
"tiobe_complexity": 2.4624855713736054,
"tiobe_coverage": 100.0,
"tiobe_duplication": 100.0,
"tiobe_fanout": 90.0,
"tiobe_functional": 100.0,
"tiobe_security": 100.0,
"tiobe_standard": 100.0

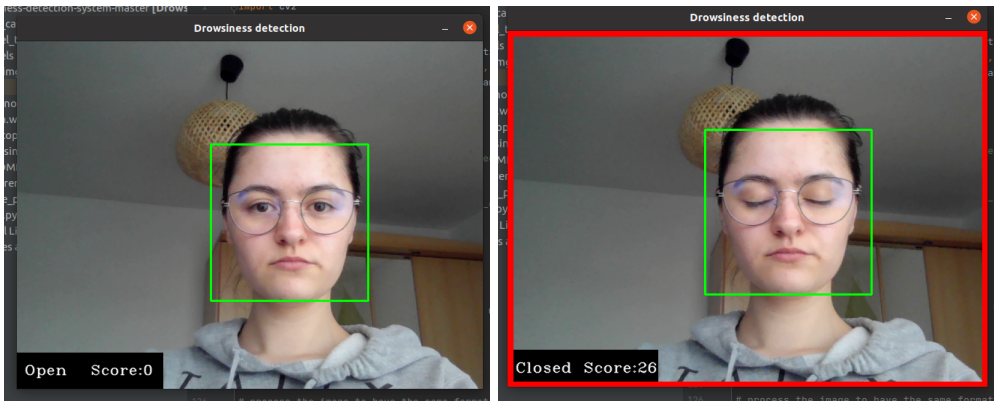
After taking a closer look at the results provided by the tool, we can see that even though the application is small (it has only 304 lines of code), the cyclomatic complexity indicates that there is at least one method that is too complex. Regarding the number of classes, methods, and attributes, they are all small, proportional to the size of the actual application. With that being said, and keeping in mind that the functionality grows together with all of these numbers, I would say that my application's functionality is moderate.

Portability

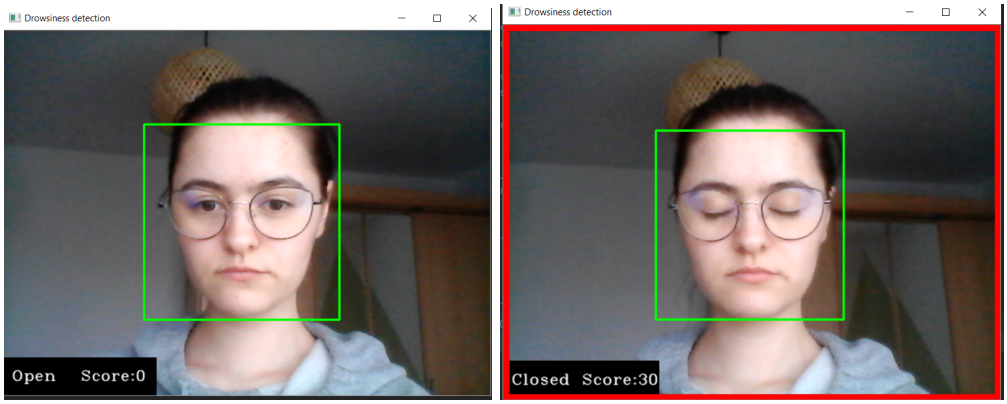
It defines the effort required to transfer a program from one hardware configuration and/or software system environment to another.

- **adaptability:** measures the degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software, or other operational or usage environments
- **installability:** measures the degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment
- **replaceability:** measures the degree to which a product can replace another specified software product for the same purpose in the same environment.

To cover the adaptability sub-characteristic, I decided to do an empirical experiment and test out my application on different operating systems, more specifically the ones that I had access to, which were Windows 10 and Ubuntu 20.04. Below you can see some screenshots of the application running on the two environments, and as expected, the program was not affected by this change in any sense.



Ubuntu



Windows

For the installability sub-characteristic, I used the checklist presented below. From the empirical test performed previously, we can say that the software is capable to run on different operating systems without any trouble, and for its installation, no other prerequisites are needed, although for the application to actually run, there is the need for a web camera connected or built in the machine we are using.

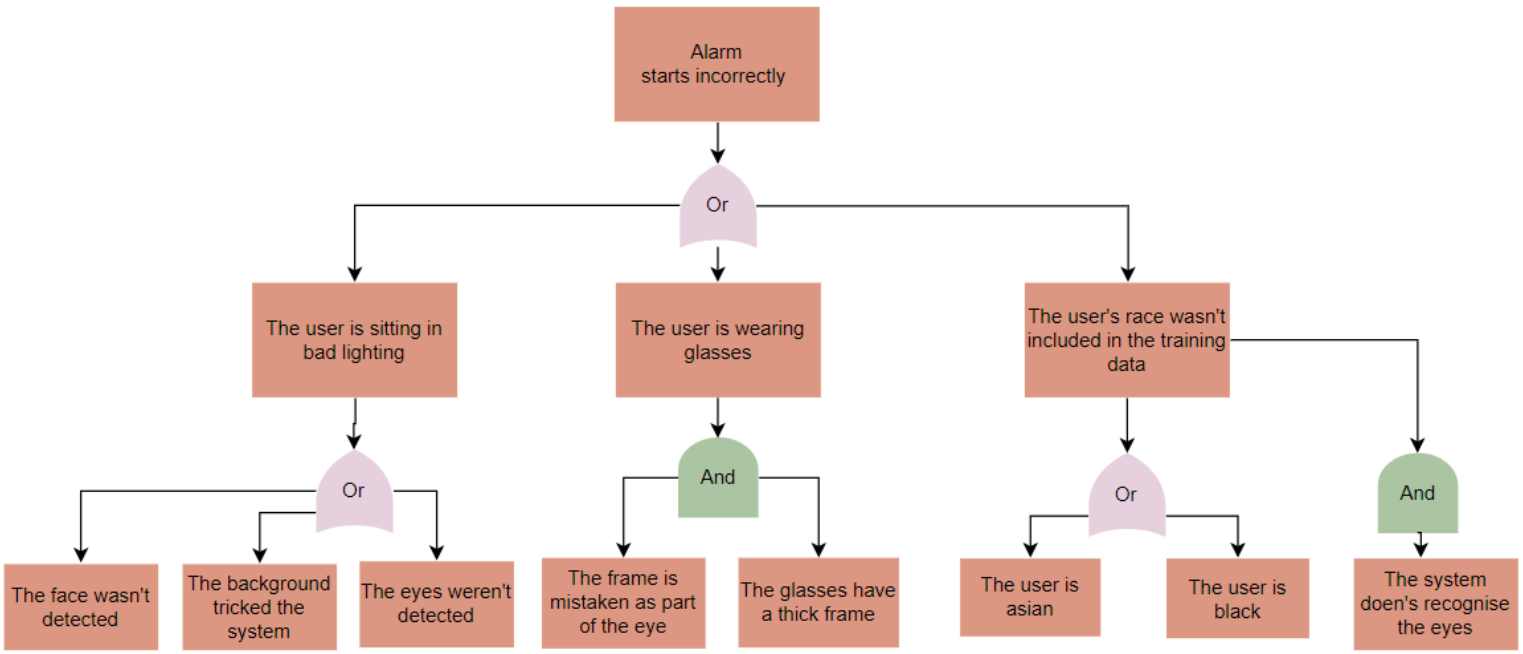
Installability checklist			
Question	Weight	Counting rule	Answer
Can the software run on any operating system?	3	0: Yes 1: No	0
How many prerequisites (on top of OS) are needed in order to install the application?	3	0: None 0.5: <=3 1: >3	0
Does the system provide an installation manual?	4	0: Yes 1: No	1
How many installation steps are required?	4	0: <=10 1: >10	0
Total = 4/14			

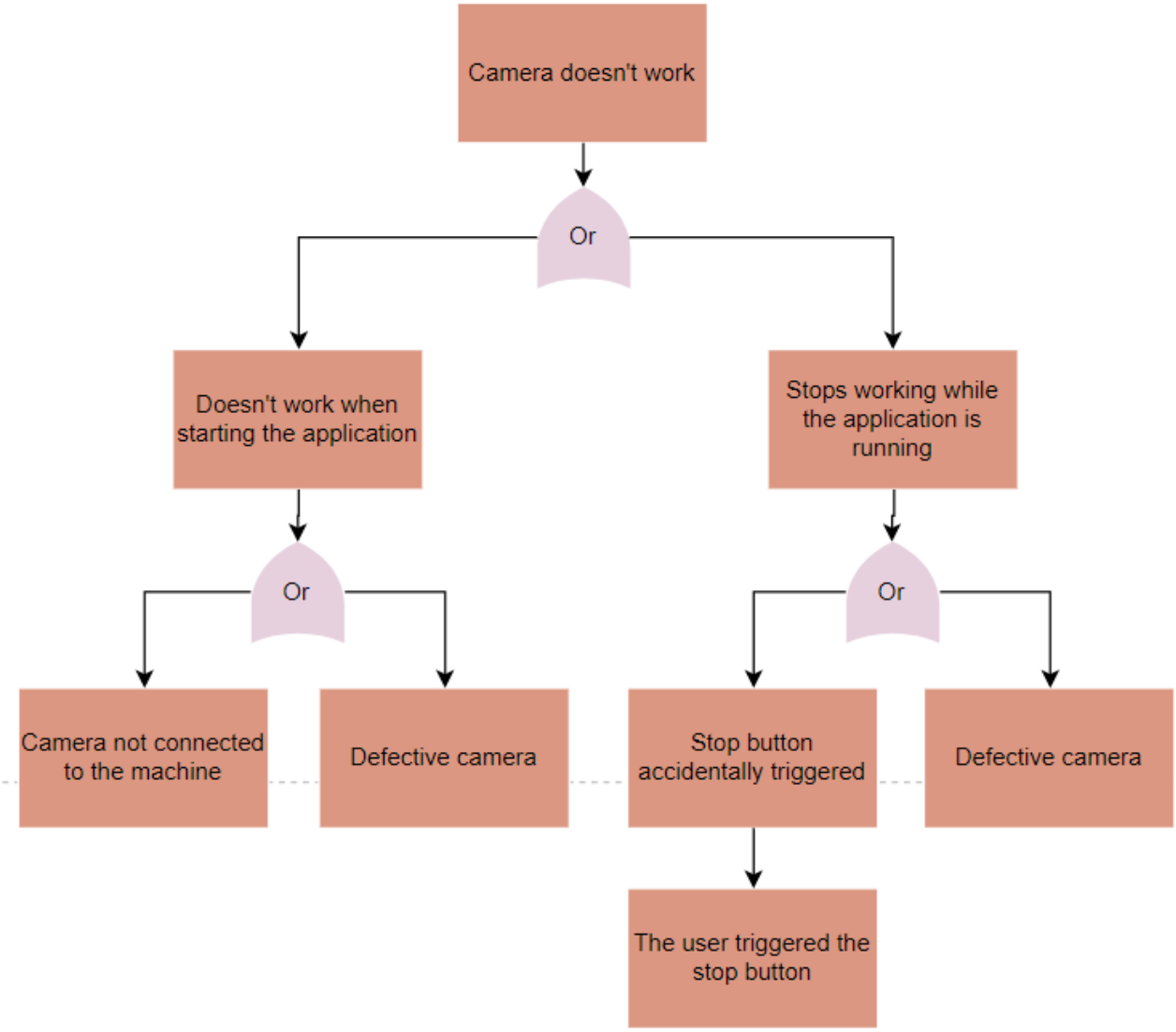
Reliability

It measures the ability of a product or component to continue to perform its intended role over a period of time to pre-defined conditions; it is measured in terms of the *mean time between failures*, the *mean time to repair*, the *mean time to recover*, the *probability of failure* and the *general availability* of the system.

- **maturity:** represents the absence of software faults that lead to failures
- **robustness / fault-tolerance:** measures the performance to specification of the system, despite some possible faults
- **recoverability:** represents the behavior of the operation after a failure

For this factor, I decided to used Fault Tree Analysis in order to gain a better understanding of the possible scenarios and errors that might infiltrate in the flow of the system.





After taking a closer look at these possible failure scenarios, it can be cernainly said that the system can encounter some serious problems when being faced with bad lighting or data that hasn't seen before like frame glasses or different eye shapes, while also being prone to hardware malfunctions like camera failure.

References:

[1]. J. Lenhard, S. Harrer and G. Wirtz, "Measuring the Installability of Service Orchestrations Using the Square Method," 2013 IEEE 6th International Conference on Service-Oriented Computing and Applications, 2013, pp. 118-125, doi: 10.1109/SOCA.2013.30.

[2]. Software Quality ISO Standards, http://www.arisa.se/compendium/node6.html

[3]. Krzysztof Sacha, “An Approach to the Evaluation of Software Quality”, 2005, Warsaw University of Technology, Nowowiejska 15/19 00-665 Warszawa, Poland