Software Engineering and basic QA in python

Carlos Tavares

December 10, 2024

Software Engineering - "The processes, activities and tools that lead to the delivery of software."

The main objective: Deliver correct software on time!

Programming vs Software Engineering

The former concerns the writing of correct code that fits into a specification. The latter includes the wide range of activities, mostly involving people (developers and clients) to make very complex programs come to life with quality.

Software engineering sub-processes i

There are five main activities in a software process:

1. Requirement elicitation;
2. Software architecture and design;
3. Development;
4. Testing and quality assurance;
5. Deployment;

The **software process**: the way all activities to develop software have to be undertaken: tools, methods, documentation, mandatory guidelines to code writing, etc.

Software engineering sub-processes ii

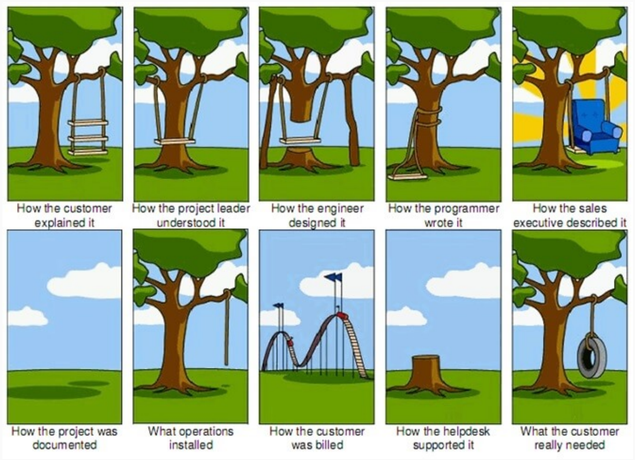
The exact shape of these processes varies with the type of software under development:

- Specific software for a specific client (service);
- Product targeting a certain market, e..g facebook, instagram;
- Specific software for industrial applications:
    - Automotive industry;
    - Nuclear powerplants;
    - Avionics.

Furthermore, software can be **critical** or **non-critical**.

Requirement elicitation i

In the case of software for a specific client:


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Requirement elicitation ii

In the case of **software products**:

Users are an entire market. Requirements eliciation require market and economic analysis.

In of specific **critical areas**, such as avionics, nunclear energy, etc:

The requirements are already very technical and defined by domain specialists.

Requirement elicitation iii

Different areas have different needs, leading to different processes: different enphasis in different phases of the software process, different tools and methods. The activities don't follow exactly the order $1 - 2 - 3 - 4 - 5$.

- **Waterfall:** the strict order $1 - 2 - 3 - 4 - 5$
- **Agile:** every activity is done in a spiral way.

Waterfall - Targets systems with steady but complicated requirements;
Agile - Targets systems where requirements change a lot.

Quality control i

Non-critical software

- Tests (system, components, unitary tests)
- Staging environments

Critical software

- The process is quite rigorous, besides tests, formal methods can be employed;
- There are standardization bodies that certify the software: **IEC 60880** for nuclear power stations software, FAA for aviation.

Software for avionics i

The software process standard for avionic systems is defined in the **DO-178C** standard;

For ground systems the **DO-278A** standard is the relevant one;

These documents determine guidelines of how to prepare and present the case for the certification with a regulatory authority: how to collect and present evidence.

Software components have to be certified by regulatory autohorities: Federal Aviation Administration (FAA) in the United States, European Union Aviation Safety Agency (EASA).

Software for avionics ii

Some interesting facts about the DO-178C:

Costs of developing software according to this standard are 10 times more than normal software!

Huge investment in vertification: 60 % to 80 % of the project budget is spent on quality assurance:

Huge documentation: 100 lines of documentation for each line of code. Developers are not encouraged to write a lot of code.

Software for avionics iii

Zero tolerance to **error**. Space Shuttle had 400 000 lines of code and a ratio of only 1 error per release.

Redudancy of software systems: there are more than one version of software doing the same thing. When conflicts arrive systems vote.

Avionics is not keen on adopting new technologies. They prefer stable and canonic technology.

Planes are mostly autonomous today! They can land and takeoff autonomously.

QA in Python

Exceptions i

Programs may generate problems in runtime. For instance:

```
x = int (input ("Write a number: "))
y = 1/x
print ("Y: ", y)
```

Will raise an error if x is zero, and the program will be interrupted.

Exceptions ii

However errors are also objects in python and they can be "catched" by try... except blocks

```
try:
      x = int (input ("Write a number: "))
      y = 1/x
      print ("Y: ", y)
except:
      print("Some error was raised here. Write some decent input
next time?")
```

Exceptions iii

Python can raise errors for many reasons:

- **SyntaxError**
- **IndexError**;
- **SystemError**;
- **TypeError**;
- **RuntimeError**
- **KeyboardError**

Exceptions iv

Exception catching can be typed to a specific exception

```
try:
    x = int (input ("Write a number: "))
    y = 1/x
    print ("Y: ", y)
except ZeroDivisionError:
    print("Division by zero")
except:
    print("Some other error happened")
else:
    print("Everything went well")
```

Exceptions v

```
try:
    x = int (input ("Write a number: "))
    y = 1/x
    print ("Y: ", y)
except ZeroDivisionError:
    print("Division by zero")
except:
    print("Some other error happened")
else:
    print("Everything went well")
finally:
    print("Regardless of what happens this block will always be
executed")
```

Ariane 5 Rocket explosion - Rocket explodes due to a non catched overflow exception;

Knight Capital Trading Glitch (2012): A software deployment error caused Knight Capital, a financial services firm, to lose over 440 million in just 45 minutes. An obsolete function failure wasn't properly handled and started selling cheap and buying expensive;

Raising exceptions i

One may be interested in signalling wrong situations in a program.
One can raise exceptions, through the instruction **raise**. Example:

```
x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

Raising exceptions ii

Another example:

```
x = "hello"

if not type(x) is int:
    raise TypeError("Only integers are allowed")
```

The assert instruction i

Another way of throwing exceptions if certain conditions are not met is through the **assert** instruction:

assert condition:

or

assert condition, message.

When a condition is not met an AssertionException is raised. It can also be used with a message

The assert instruction ii

Example:

```
x = 10

assert x < 9, "This number is greater than 9"
```

The use of Exceptions and asserts

It is clear that a good management of exceptions is a paramout in real world software, i..e it is essential to the fault tolerance of software.

It can also be used to debug and test programs:

- Identify errors in programs;
- Debug track and correct the errors in programs

Debugging

Develop a program to calculate to:

- Receive a number between 20 and 40;
- Retrieve its both digits;
- Sum the year of your birth to each digit;
- Multiply for 46^2
- Sum them modulo 389
- Divide by 10;
- Present it to the user.

Testing

Unit tests are the simplest tests that are made in software, and usually are implemented by the programmer. They aim at testing small bits of code, with well defined aims.

Questions?