WHY TESTING IS IMPORTANT

Software testing has been an essential part of software engineering as long as we have called it by that name. Here we will discuss the history of it and look at how we have evolved since.

TRAYPORT

A TMX COMPANY

CONTENTS

CHAPTER

1

A BRIEF HISTORY OF SOFTWARE TESTING

A BRIEF HISTORY OF SOFTWARE TESTING

We have all encountered bugs in software during our lives, whether it be our desktops, phones or smart devices from your car to your washing machine. The brief history of computing and software is short in the grand scheme of technology, yet from the moment software engineering as a term was coined, we were already acutely aware of the need for testing within software.

The above occurred in 1968 at a NATO conference on software engineering that had many leading computer science figures in participation. During this conference, both testing and testing automation were discussed, and this was a time when hardware was still considered more crucial to the machine than software.

```
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = [illegible]
Category.delete_all; Category [illegible]
Shoulda::Matchers.configure do [illegible]
  config.integrate do [illegible]
    with.test_framework [illegible]
    with.library :rails
  end
end

# Add additional requires [illegible]

# Requires supporting [illegible]
# spec/support/ and its [illegible]
# run as spec files by [illegible]
# in _spec.rb will both be [illegible]
# run twice. It is [illegible]
# end with _spec.rb. [illegible]
```

No results found for 'mongoid'

In the software field, we tend to forget our past and have to re-invent the wheel far too often. This is apparent in the concepts of testing that were already discussed in 1968 but even to this day sometimes elude people:

- A software system can best be designed if testing is interlaced with the designing instead of being used after the design.
- System testing should be automated as well. A collection of executable programs should be produced and maintained to exercise all parts of the system.
- The proper testing of large programming systems is virtually impossible; but with sufficient resources, enough testing can be performed to allow a good evaluation to be made.
- Seemingly, quite minor changes may have a profound effect upon the operation of a package.
- "...if the users are convinced that if catastrophes occur the system will come up again shortly, and if the responses of the system are quick enough to allow them to recover from random errors quickly, then they are fairly comfortable with what is essentially an unreliable system."

SOURCE: NATO Software Engineering Conference, 1968

CHAPTER

2

COMPLEXITY & RELIABILITY

COMPLEXITY & RELIABILITY

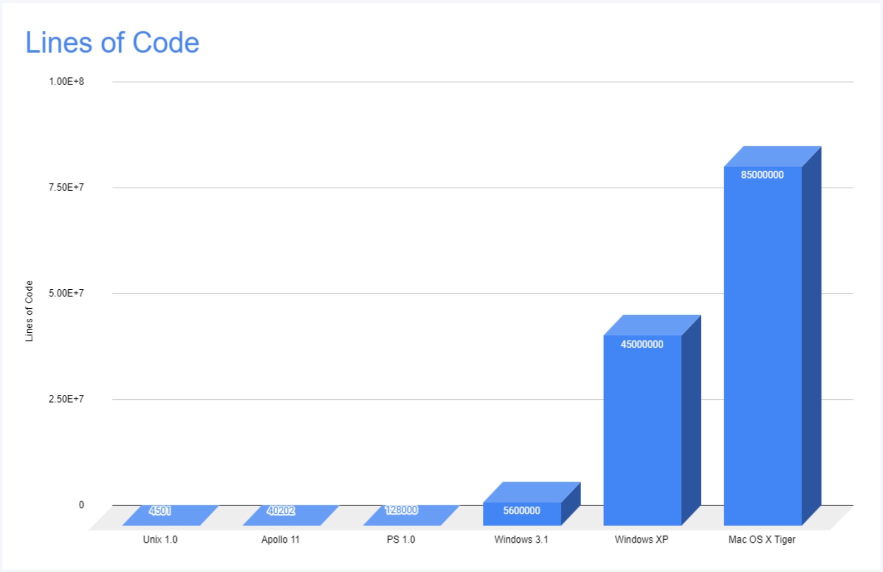
In 1968 machines and software were far simpler compared to modern computers and smart devices, yet even then it was noted:

- Complexity has been shown to be linked to reliability.
- Any attempt to test exhaustively, such a system is going to be very difficult. It is fair to say it will be as intellectually challenging to test the complex software systems of the future as it will be to design them initially.

This is truly the correct assessment as the complexity of our software has grown exponentially. To illustrate this explosion of complexity, we can look at the lines of code in software from the past to the modern age.

As the NATO conference took place in 1968, a good example of one of the most advanced systems at the time would be the Apollo 11 moon mission. The computer system on board Apollo 11 was constructed by using 40,202 lines of code.

After this milestone, we have rapidly exploded in the size of our applications measured by lines of code, reaching 85 million lines of code at Mac OS Tiger X in 2005. That is already almost 20 years ago as of this year.



It is said that Google for instance is roughly 2 billion lines of code!

CHAPTER 3

85 MILLION LINES OF CODE

85 MILLION LINES OF CODE

To grasp how much 85 million lines of code is we can imagine it as books.

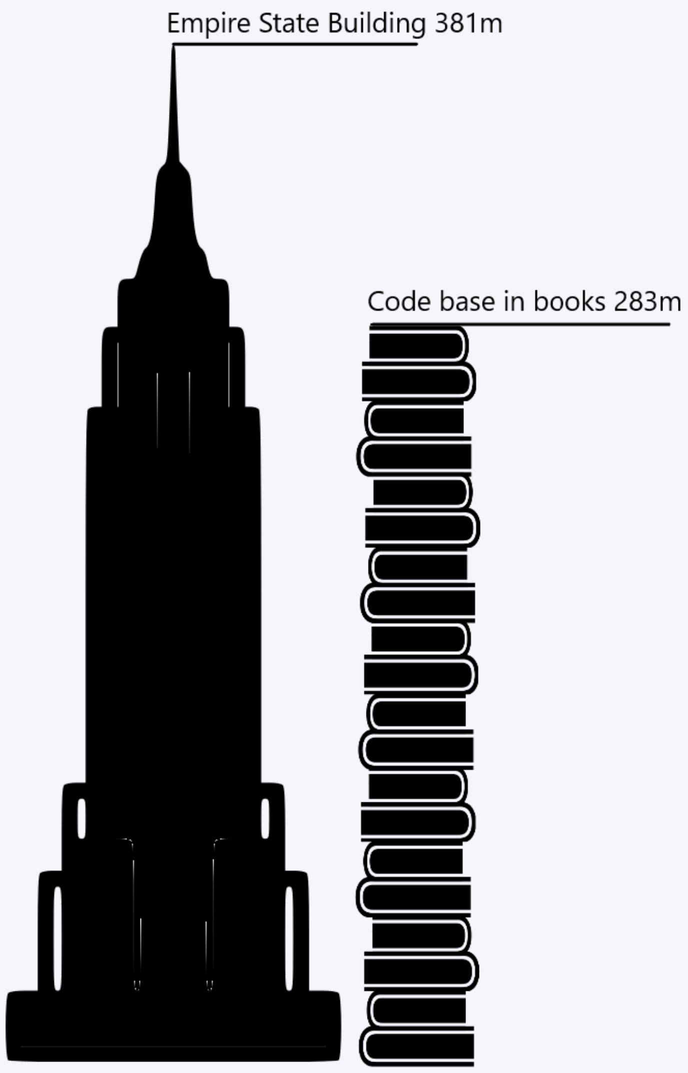
- An average book has 400 pages
- On average there are 300 words per page
- Average sentence has 20 words
- One page of a book is roughly 15 lines
- One book is roughly 6,000 lines long

If we convert this to match the 85 million lines of code we get:

~14,167 books

An average adult reads 12 books a year

- It would take ~1180 years to read the code base at a book reading speed



This is not the case, as the code base is more of a hyperlink type of wiki page.

Which you have to absorb the whole content and jump in between to understand it, as it is not a linear story from beginning to end. If you imagine this pile of books to read we can compare it to the size of Empire State Building.

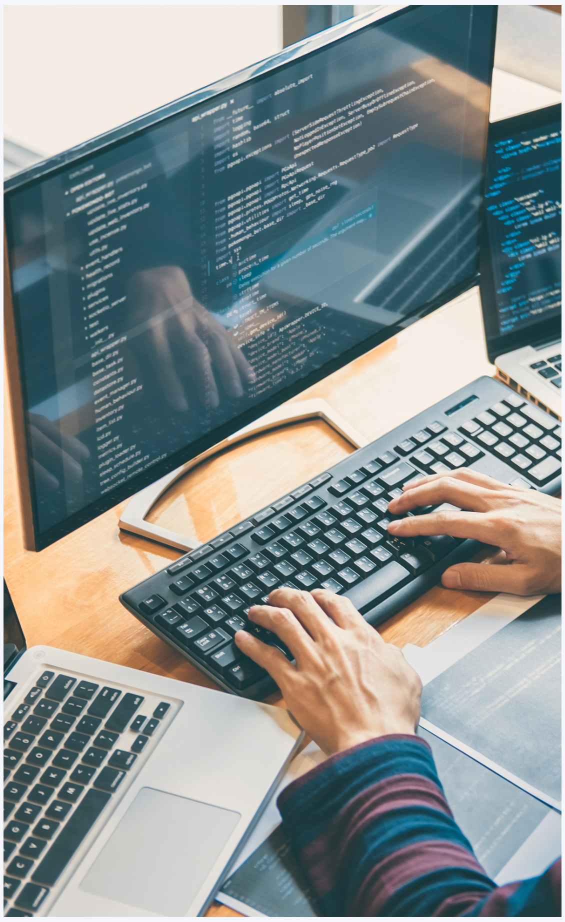
CHAPTER

4

THE PYRAMID

THE PYRAMID

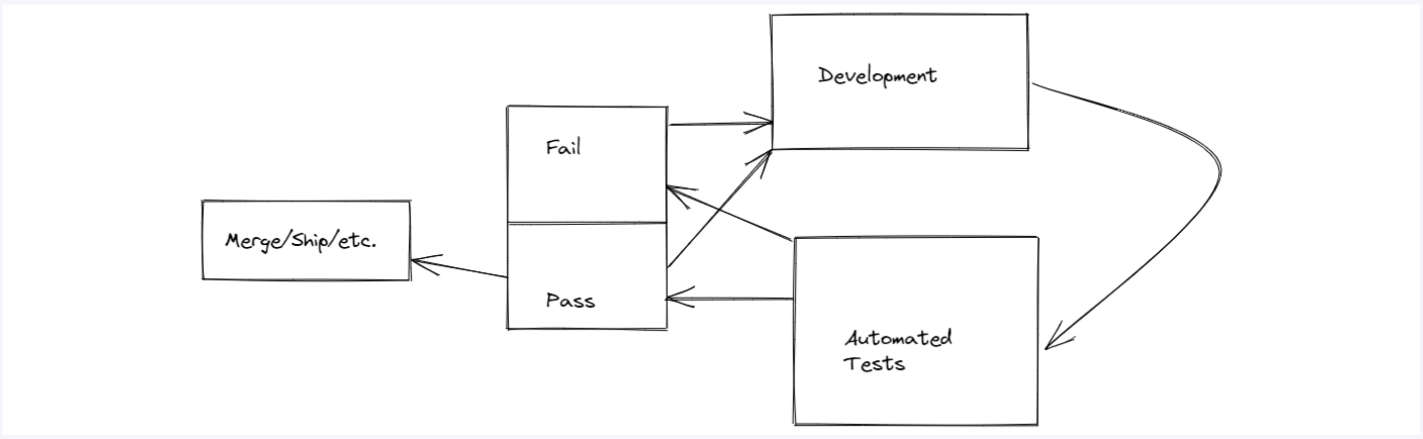
Writing 14,167 books without typos, let alone writing 85 million lines of code, error free without testing is practically impossible. Even with testing, finding all possible errors and bugs is still practically impossible and this is due to all of the added complexity that goes even deeper than just the lines of code.



Considering the testing of such magnitude, we instantly recognize that manually covering such a vast area is not feasible unless you employ a large company's worth of testers to run through manual testing every time something changes, and even then this would be incredibly arduous.

Hence the need for automated testing in modern software, where due to the complexity, it is crucial to enable a fast feedback loop for developers to assess whether their changes have broken other parts of the system. There are significant caveats in the process that often lead to more problems as they delay the feedback loop from automated testing back to the developers that should be relatively rapid.

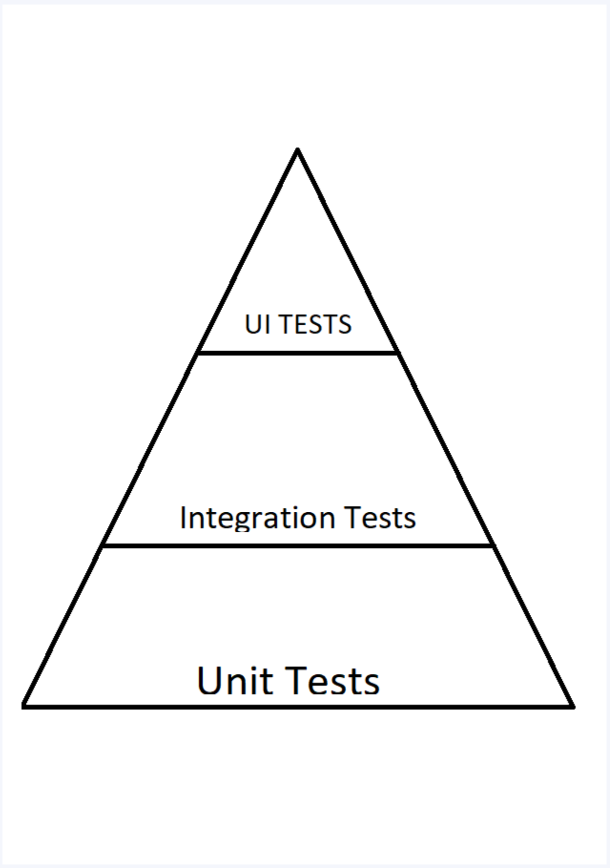
The common stumbling block around this area is when automated tests become unwieldy and slow which deteriorates the feedback loop. This typically occurs when the test distribution errors from the typical pyramid style approach.



THE PYRAMID STYLE APPROACH

The pyramid represents how the fastest type of tests should be the most and the slowest the fewest. Simply because software is rarely stale and it grows and evolves with iterations as does testing.

This leads to complexity growth in the testing phase, and in worst case scenario tests will become unmanageable and take days to run.



The constant change of software and the tests around it can be considered as entropy, "lack of order or predictability; gradual decline into disorder", that slowly creeps into software and must be managed via refactoring and general code maintenance, or else both the software and the tests fall into disorder.

This is especially true in the modern age as in the past we had physical buttons, punch cards and paper outputs of our functions that nowadays are only found in museums. Following the transition to command line, the rise of graphical user interface slowly occurred and we gained the slowest type of tests on top of the pyramid.

CHAPTER 5

THE HUMAN EXPERIENCE

THE HUMAN EXPERIENCE

Another layer of complexity is added to quality since we need to consider the user experience (UX) with modern software as well. This is something that is very subjective to the user and automating something so human seems near impossible. The human testers, however, can assess usability from various aspects while doing a variety of exploratory testing.

Exploratory testing, or chaos engineering as Netflix likes to call it, is an integral part of testing. Automated testing is an excellent tool for working as a safety net and a feedback loop, but in the insurmountable complexity of modern software and how it is almost always interconnected to various other facets, we do require a more "human" approach to control the quality as well.

Indeed an easy example of why this more human type of testing can be so efficient is how humans were able to solve protein folding problems that super computers were struggling to solve, for "That's no small task, considering that even a moderately sized protein can theoretically fold into more possible shapes than there are particles in the universe."

SOURCE:
Bohannon, John. "Video Game Helps Solve Protein Structures." Science,
https://www.science.org/content/article/video-game-helps-solve-protein-structures.

The same level of complexity, or even more so, applies to the software we are testing, and therefore a few inquisitive and capable humans can in some ways outdo super computers in the ability to detect issues in software.

ETHICAL IMPLICATIONS

Once we move past the technical aspects of testing, we ought to examine a more philosophical testing aspect that is rarely, if ever, considered but is in fact equally important.

Consider the VW emission testing defeating software that lead to the Volkswagen emissions scandal.

“The agency had found that Volkswagen had intentionally programmed turbocharged direct injection (TDI) diesel engines to activate their emissions controls only during laboratory emissions testing, which caused the vehicles’
NOx output to meet US standards during regulatory testing. However, the vehicles emitted up to 40 times more
NOx in real-world driving. Volkswagen deployed this software in about 11 million cars worldwide, including 500,000 in the United States, in model years 2009 through 2015.”

SOURCES:

"EPA, California Notify Volkswagen of Clean Air Act Violations / Carmaker allegedly used software that circumvents emissions testing for certain air pollutants". US: EPA. 18 September 2015. Archived from the original on 2 March 2017. Retrieved 1 July 2016.

Jordans, Frank (21 September 2015). "EPA: Volkswagon [sic] Thwarted Pollution Regulations For 7 Years". CBS Detroit. Associated Press. Retrieved 24 September 2015.

"Abgasaffäre: VW-Chef Müller spricht von historischer Krise". Der Spiegel. Reuters. 28 September 2015. Retrieved 28 September 2015.

Ewing, Jack (22 September 2015). "Volkswagen Says 11 Million Cars Worldwide Are Affected in Diesel Deception". The New York Times. Retrieved 22 September 2015.

This software embedded to the cars could have been the best software with no bugs, but was it good quality? Definition of quality:

- "the standard of something as measured against other things of a similar kind; the degree of excellence of something"
- "a distinctive attribute or characteristic possessed by someone or something"

SOURCE: GOOGLE OXFORD LANGUAGES

One could argue the final hurdle for testing is to assess moral and ethical implications of the software and its quality. If VW emissions defeating software was perfect technically, it surely lacked the moral and ethical grounds to be called good quality or to pass testing without raising significant amount of questions.

SUMMARY

In the last 100 years we have gone from machines with physical gears and cogs to having supercomputers in our pockets with access to most of human information within our fingertips. This leap in computer technology has come with the side effect of ever increasing complexity in software. We have reached a point where a single person is unlikely to fully grasp how a piece of software works, hence the need for validation is paramount. Integrating testing to the flow of development is important but one should not forget the more human side of testing and its implications in the context of quality.

AUTHOR

Teemu Ruuskanen
Contractor Senior Exploratory Tester