Software testing

In this train, we'll explore the vital practice of software testing and understanding its role in bug detection and confirming software alignment with business needs. We'll differentiate between verification and validation, illustrating their importance in ensuring correctness and relevance. Real-world examples will emphasise the impact of testing in preventing issues, while we discuss overarching testing goals like defect identification and user validation.

Learning objectives

By the end of this train, you should be able to:

- Define the concept of software testing and its dual role in bug detection and ensuring alignment with business requirements.
- Differentiate between verification and validation processes in software testing.
- Understand the real-world impact of software bugs by analysing illustrative examples.
- Grasp the overarching goals of software testing, including defect identification and user-centric validation.

Outline

1. What is software testing?
2. Why is software testing important?
3. The goals of testing.
4. The levels of testing.

What is software testing?

Software testing is a set of **activities which are intended to find bugs within software**, as well as validate and verify the software meets the business requirements and needs as designed.

Importantly, a *bug*, or a defect within the software, is an error that causes an **incorrect or unintended result.**

Software testing techniques can be roughly classified into either verification or validation processes.

Verification

During verification, we test whether the software was built according to the **design specification outlined.** Ultimately, we are verifying whether the developer completed the development as it was envisioned.

In this sense, verification tests ask:

â€œAre we building it right?â€

Validation

During validation, we test that the new software **solves for the original business problem,** or fulfils the original business need.

In this sense, validation tests ask:

â€œAre we building the right thing?â€

Why is software testing important?

When bugs in software go undetected and are released in live code (known as production code), they can produce serious consequences:

- Loss of revenue and increased costs.
- User frustration and application abandonment.
- Prolonged project delivery and missed milestones.

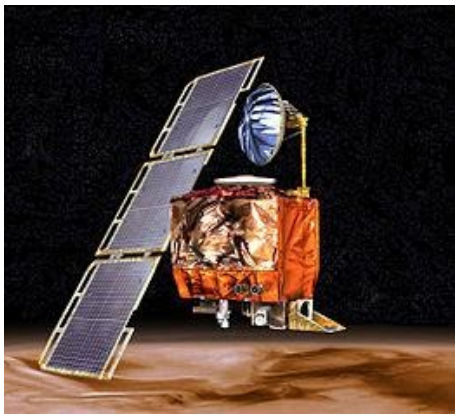
â€¦and in some cases, they can cause loss of lives!

Let's look at a few examples of software bugs to illustrate why it is important to test our software.

Mars Climate Orbiter

The Mars Climate Orbiter was launched in 1998 to study Mars climate changes. Unfortunately, the developer made a simple mistake and used English instead of metric units to coordinate the craft's flight path. This bug caused it to get too close to the Mars atmosphere, which caused the Orbiter to disintegrate.

The net loss for this error was estimated to be $327 million!



Prisoners released early

Another example surrounds a software system used in the United States to calculate the remaining term which prisoners had to serve based on their behaviour. The system was found to have had a bug, introduced in an update in 2002, which caused this term to be erroneously calculated. It was only detected in 2015 though â€“ causing over 3,200 prisoners to be released early from jail!

Oscars system design error

The story of how the Oscars messed up in their awarding of the 2017 'Best Picture' award is a very interesting take on poor system design choices.

Listen to an interesting podcast on this blunder at [Cautionary Tales](https://99percentinvisible.org/episode/cautionary-tales/).

The goals of testing

In light of the horror stories above (there are many more of these!) the goals of testing are to find bugs before software is published; to verify and validate the system.

The goals for verification testing include:

- To show the presence of defects.
- To reduce the probability of bugs remaining in software at the time of launch.

Importantly, one of the goals for verification is **not** to make the software *bug-free*. This is not feasible without large time and financial investment.

The goals for validation testing include:

- To confirm that intended users can accurately use the software.
- To ensure the end result solves the original business problem.

The levels of testing

There are four primary levels in software testing â€“ each concerned with testing the software at different levels of granularity:

1. Unit testing
2. Integration testing
3. System testing
4. Acceptance testing

Unit testing

Unit tests ensure that small fundamental pieces of code, known as units, behave correctly when invoked in isolation. Examples of code units include functions or class methods and are defined with clear parameters and scope.

Here's another story to illustrate why unit testing is so important: In 1994 shortly after Intel introduced its Pentium microprocessor, it was found to perform some calculations incorrectly. It was discovered that a table was not being fully initialised and a few cells contained zero instead of 2.

Intel claimed to have run millions of tests â€œusing this tableâ€, however, these were probably system tests.

A unit test would have revealed that the initialisation loop was ending too early.

Intel had to recall the chip at the cost of millions of dollars.

Related to unit testing is the process of module testing. Several units of code make up a module, which often has a larger area of responsibility than a single function would. For example, a single Python script which we run in order to perform a task could be considered as a software module. Here it is important to note that after unit testing is performed, we still need to validate that the units *interact* with each other correctly. This intermediate testing level is known as module testing.

Integration testing

In real-world software, several modules are integrated into a larger system. Integration testing is to module testing what module testing is to unit testing.

It is designed to assess how modules interface with one another:

- Do they have consistent assumptions? e.g. Are the variables they share all of a compatible type?
- Do they communicate correctly?

Like unit and module testing, integration testing is done by programmers.

System testing and acceptance

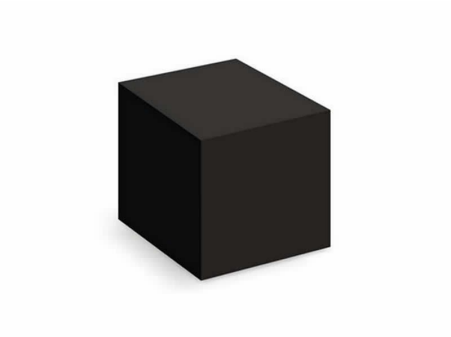
System testing is designed to determine whether the integrated system meets its specification. It assumes the pieces work individually and interface correctly. Importantly, system testing is done by a separate testing team.

Acceptance testing is designed to determine whether the completed software meets usersâ€™ needs. It must involve users or people with strong domain knowledge. In this sense, once again acceptance testing is done by a separate testing team.

Looking at these two testing levels together, we can see that:

- System testing corresponds to verification: Did we build the system right?
- Acceptance testing corresponds to validation: Did we build the right system?

Glass and Black Boxes



Related to the various levels of testing, an important concept of test implementation surrounds the testing paradigm:

- If test cases are created based only on an item's problem specification, with no knowledge of the underlying design/implementation of its code being assumed, this process is known as Behavioral or **Black Box testing**.
- Conversely, if test cases assume access to the structure, logic, and implementation code for an item, this process is known as White o**Glass Box testing**.

Summary

In this train, we introduced the concept of software testing, emphasising its crucial role in bug detection and ensuring alignment with business requirements. It explores the distinctions between verification and validation processes, highlighting their significance in confirming correctness and relevance. Real-world examples, such as the Mars Climate Orbiter and a software system leading to the early release of prisoners, underscore the potential consequences of undetected bugs. The goals of testing are outlined, focusing on defect identification and user-centric validation. The train also delves into the four primary levels of software testing â€“ unit testing, integration testing, system testing, and acceptance testing â€“ illustrating their importance in ensuring the correct behaviour of software components at different levels of granularity. Additionally, it introduces the concept of Black Box testing paradigms based on the level of knowledge about the underlying code during test case creation.