TESTING PRINCIPLES

Seven principles

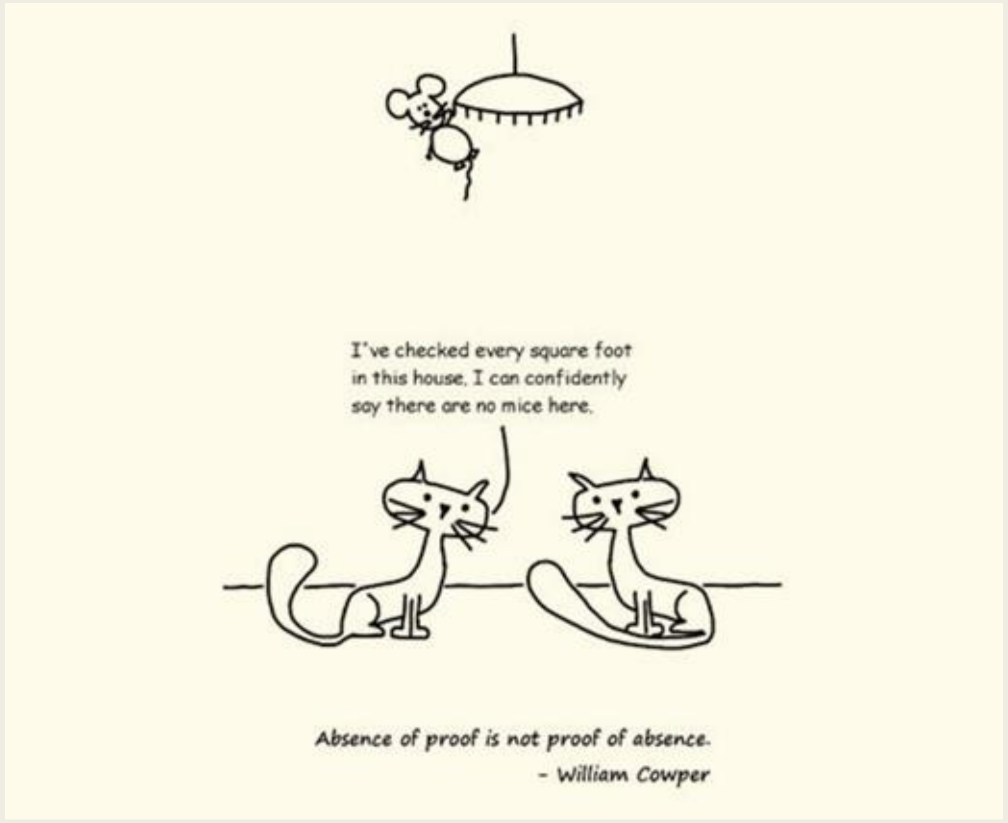
7 Principles of Software Testing

1. Testing shows presence of defects
2. Exhaustive testing is not possible
3. Early testing
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of errors fallacy

1) Testing shows a presence of defects

Testing talks about the presence of defects and don’t talk about the absence of defects.

The testing process does not guarantee that software is 100% error-free. Even if testers cannot find defects after repeating testing, it does not mean the software is 100 % bug-free.


I’ve checked every square foot in this house. I can confidently say there are no mice here.
Absence of proof is not proof of absence.
- William Cowper

2) Exhaustive testing is not possible

It is impossible to test EVERYTHING – all combinations of inputs and preconditions.

Example 1: The password field accepts all characters in lower and upper case, numbers and all special characters.

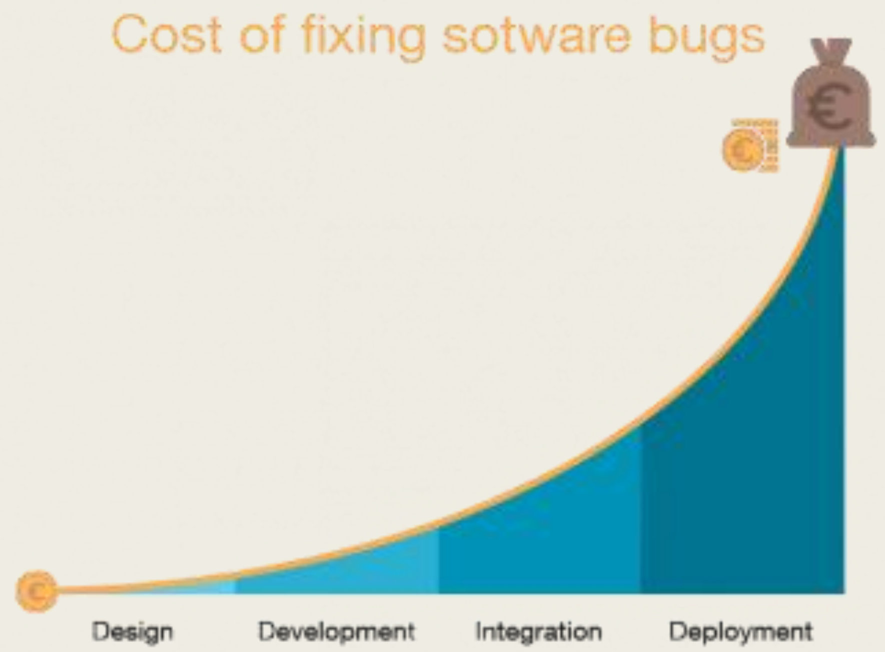
Example 2: The fire system will be activated if the temperature is greater than 30.

Cont’d

3) Early Testing

Testing should start as early as possible in the Software Development Life Cycle.

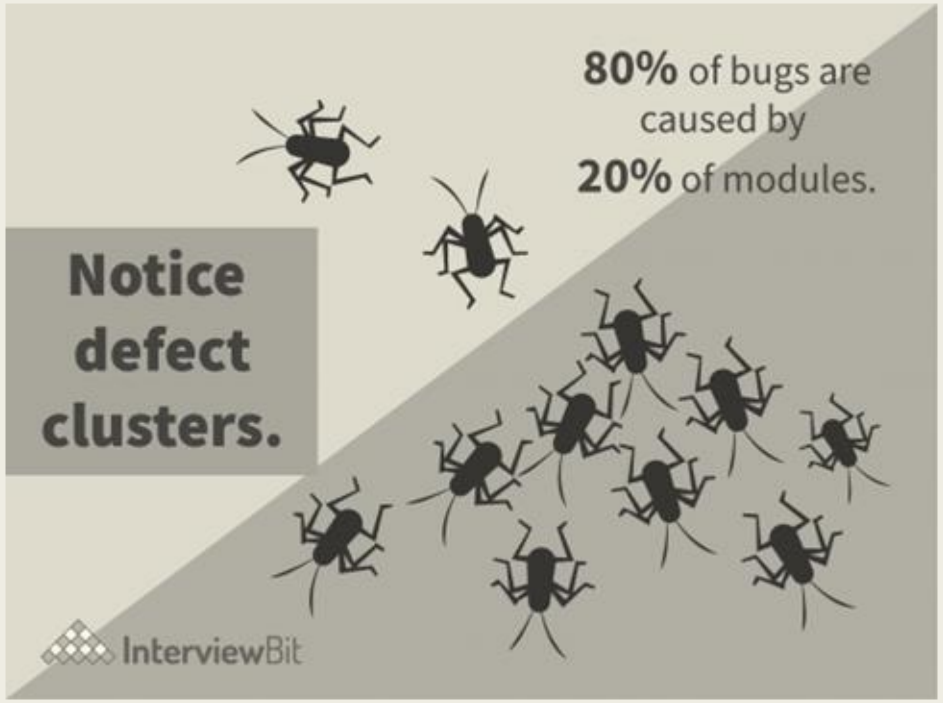
The cost to fix a bug increases exponentially with time as the development life cycle progresses.
Let’s consider two scenarios. In the first case, you found an incorrect requirement in the requirement gathering phase. In the second case, you found a defect in a fully developed functionality. It is less expensive to fix the incorrect requirement than fully developed functionality that isn’t working the way it should



4) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected.

Pareto Principle (80-20 Rule) states that 80% of issues originate from 20% of modules, while the remaining 20% originate from the remaining 80% of modules.

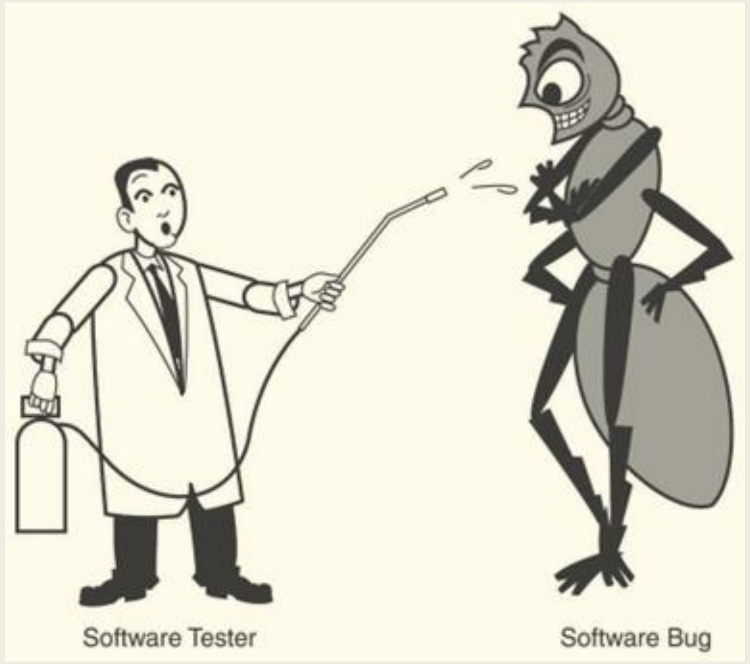


5) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects.

This means the practice of repeating the exact same test cases over and over again. As time passes, these test cases will cease to find new bugs. Developers will create tests which are passing so they can forget about negative or edge cases.

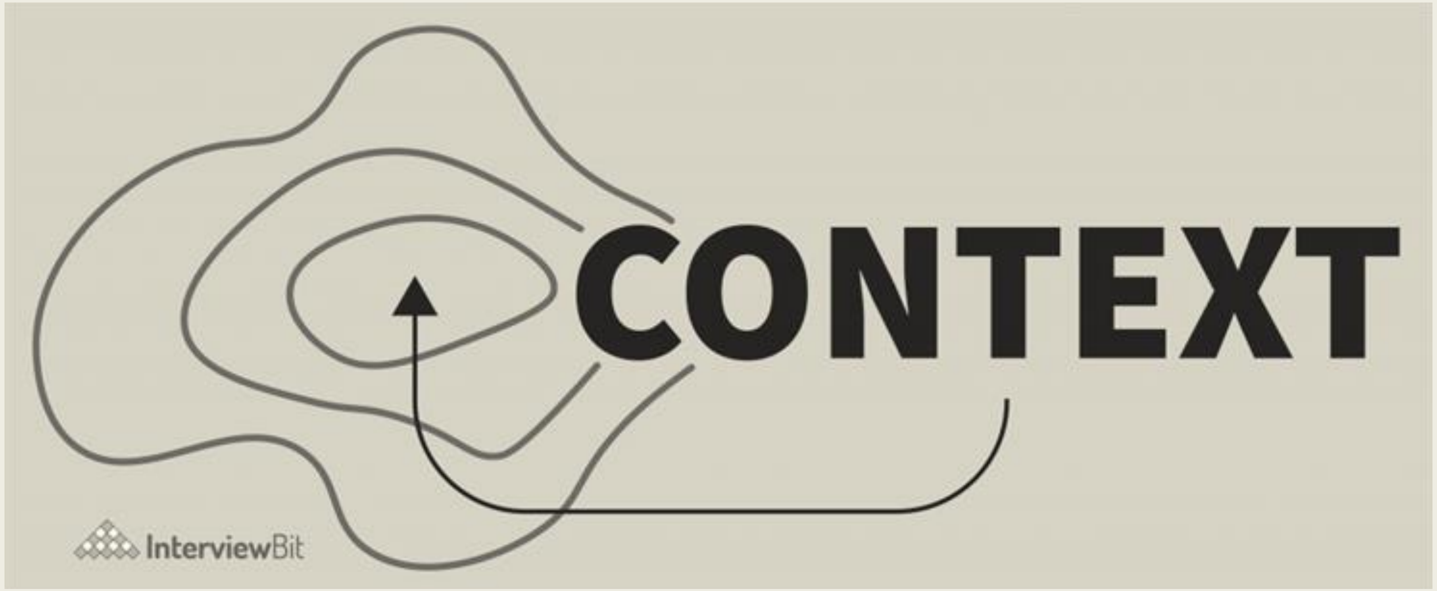
The test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.



6) Testing is context dependent

Each type of software system is tested differently.

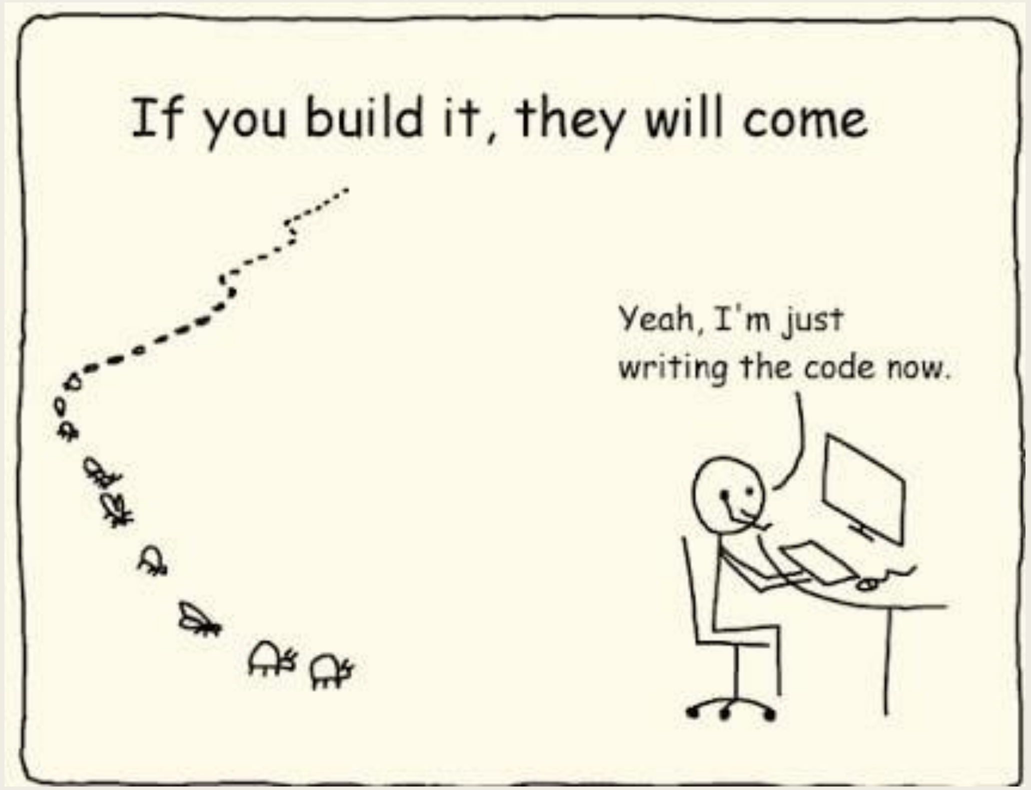
Various methodologies, techniques, and types of testing are used depending on the nature of an application. For example, health industry applications require more testing than gaming applications, safety-critical systems (such as an automotive or airplane ECU) require more testing than company presentation websites, and online banking applications will require different testing approaches than e-commerce sites or advertising sites.



7) Absence of Error – fallacy

Each type of It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. system is tested differently.

In order for software systems to be usable, it must not only be 99% bug-free software but also fulfill the business needs and user requirements.


If you build it, they will come
Yeah, I'm just writing the code now.

Conclusion:

Applying these thoughtful principles to your testing can help you become more efficient and focused and can even help improve the quality of your overall testing strategy.

You’ll often find that applying one principle will result in other principles naturally falling into place. Early testing, for example, can help mitigate the “absence of errors fallacy” - incorporating testers at the requirements stage can help ensure the software meets client expectations/needs. Combining all these principles can help you utilize your time and effort efficiently and effectively.

Thanks..

Yomna Ragab