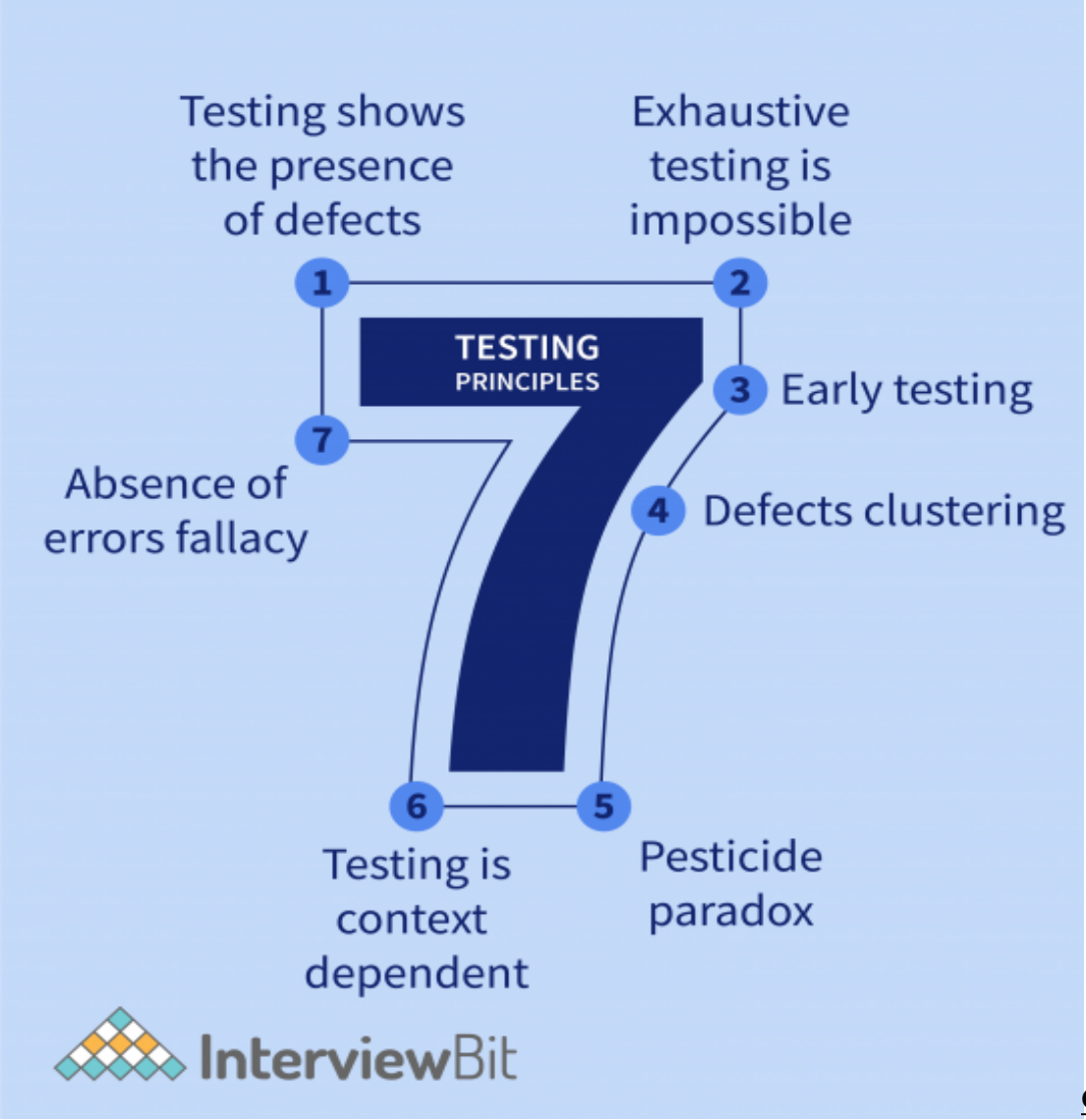
PRINCIPLES OF SOFTWARE TESTING



G

1. Testing Shows the Presence of Defects

As stated in this testing principle, “Testing talks about the presence of defects and doesn’t talk about the absence of defects”. In software testing, we look for bugs to be fixed before we deploy systems to live environments – this gives us

confidence that our systems will work correctly when goes live to users. Despite this, the testing process does not guarantee that the software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.

Even if testers cannot find defects after repeating regression testing, it does not mean the software is 100 % bug-free. For instance, an application may appear to be error-free after passing various stages of testing, but when it is deployed in the environment, an unexpected defect can be found. Team members should always adhere to this concept, and effort should be made to manage client expectations.

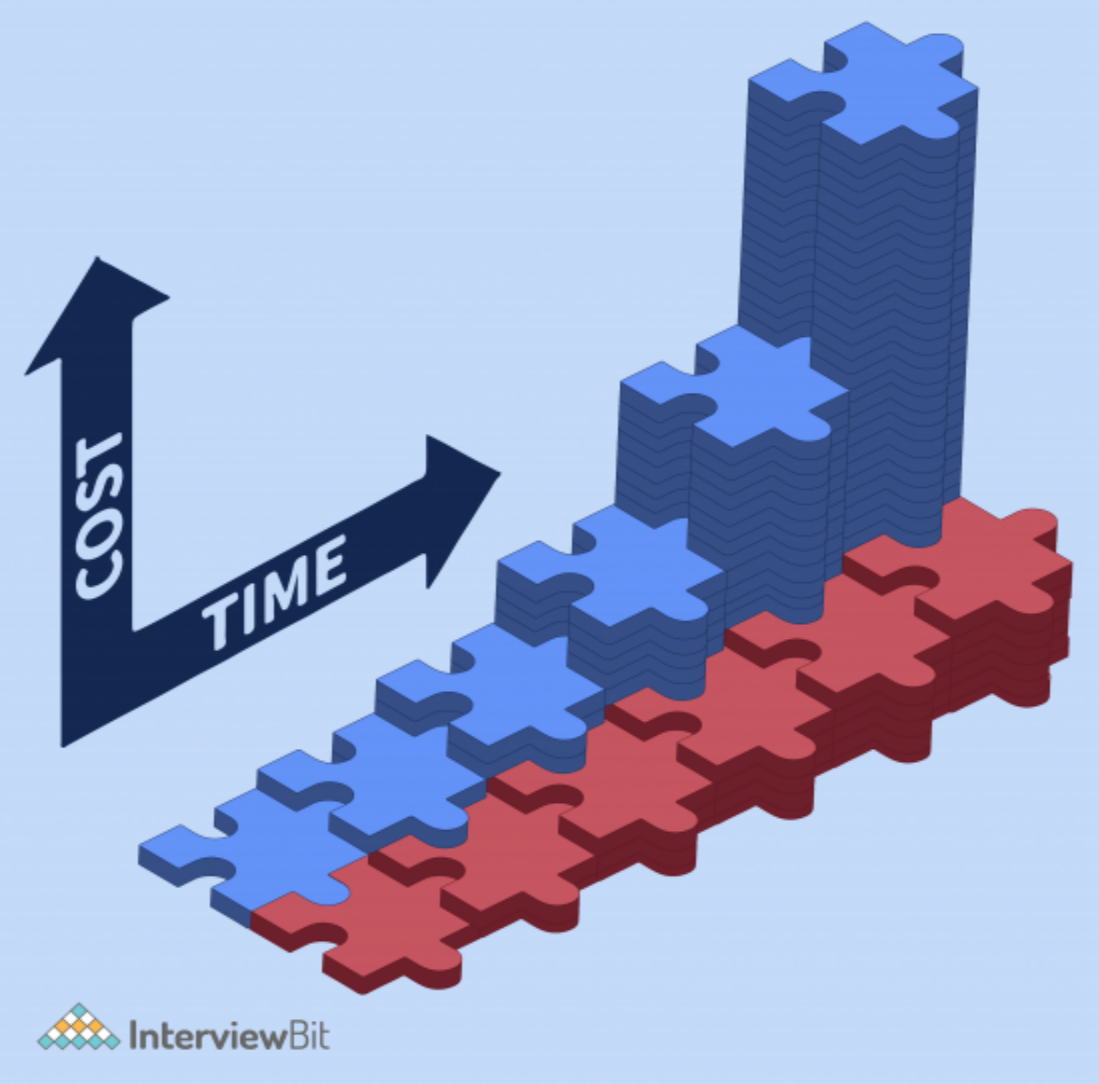
2. Exhaustive Testing is Impossible

Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions. No matter how hard you try, testing EVERYTHING is pretty much impossible. The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

Consider the case when we have to test an input field that accepts percentages between 50 and 55, so we test the field using 50, 51, 52, 53, 54, 55. Assuming that the same input field accepts values from 50 to 100, we will need to test using 50, 51, 52, 53, …., 99, 100. This is a basic example. You may think that an automation tool would be able to accomplish this. But imagine a field that accepts a billion values. Will it be possible to test all possible values?

As long as we continue to test all possible scenarios, the software execution time and cost will increase. In order to avoid doing exhaustive testing, we will take into consideration some important testing criteria effects such as risks and priorities as part of our testing efforts and estimates.

3. Early Testing



1. Testing Shows the Presence of Defects

As stated in this testing principle, “Testing talks about the presence of defects and doesn’t talk about the absence of defects”. In software testing, we look for bugs to be fixed before we deploy systems to live environments – this gives us confidence that our systems will work correctly when goes live to users. Despite this, the testing process does not guarantee that the software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.

Even if testers cannot find defects after repeating regression testing, it does not mean the software is 100 % bug-free. For instance, an application may appear to be error-free after passing various stages of testing, but when it is deployed in the environment, an unexpected defect can be found. Team members should always adhere to this concept, and effort should be made to manage client expectations.

2. Exhaustive Testing is Impossible

Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions. No matter how hard you try, testing EVERYTHING is pretty much impossible. The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

Consider the case when we have to test an input field that accepts percentages between 50 and 55, so we test the field using 50, 51, 52, 53, 54, 55. Assuming that the same input field accepts values from 50 to 100, we will need to test using 50, 51, 52, 53, …., 99, 100. This is a basic example. You may think that an automation tool would be able to accomplish this. But imagine a field that accepts a billion values. Will it be possible to test all possible values?

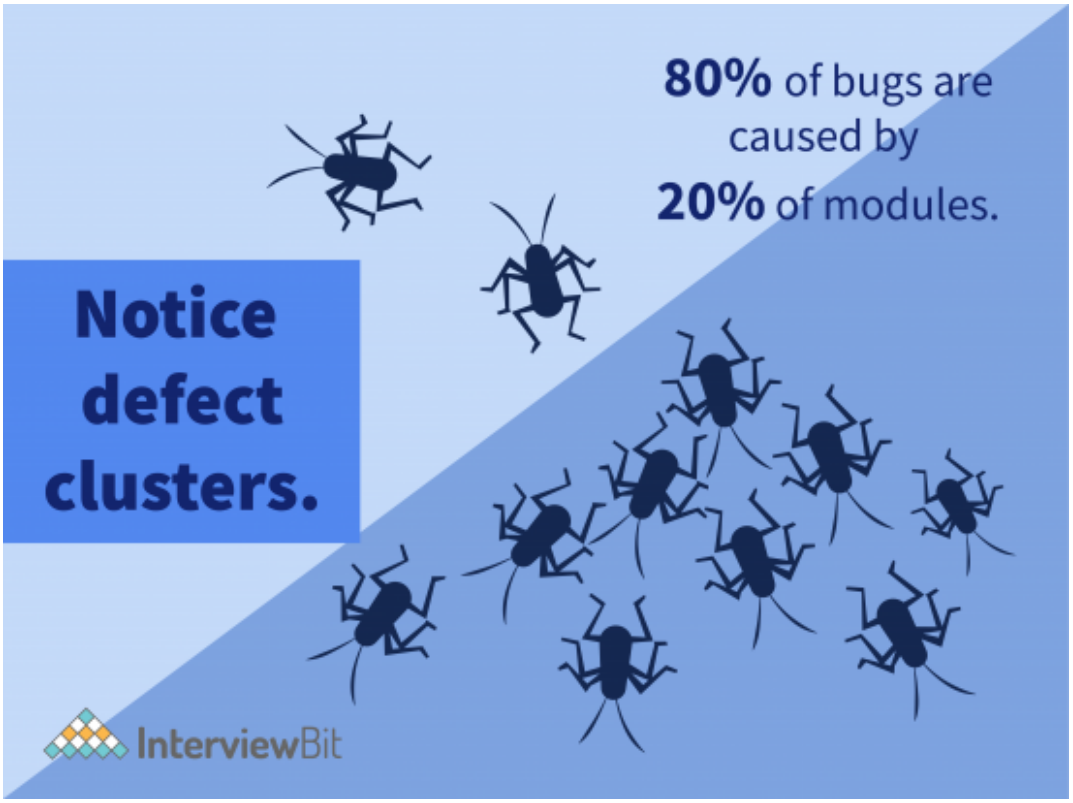
As long as we continue to test all possible scenarios, the software execution time and cost will increase. In order to avoid doing exhaustive testing, we will take into consideration some important testing criteria effects such as risks and priorities as part of our testing efforts and estimates.

3. Early Testing

In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC). For instance, testing the requirements before coding begins. Amending issues during this stage of a project's life cycle is much cheaper and easier than amending issues at the end of the project when we must write new sections of functionality, resulting in overruns and late deadlines. The cost to fix a bug increases exponentially with time as the development life cycle progresses as shown in the following figure.

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. Therefore, to improve software performance, software testing should begin at the initial phase, that is, during requirement analysis.

4. Defect Clustering

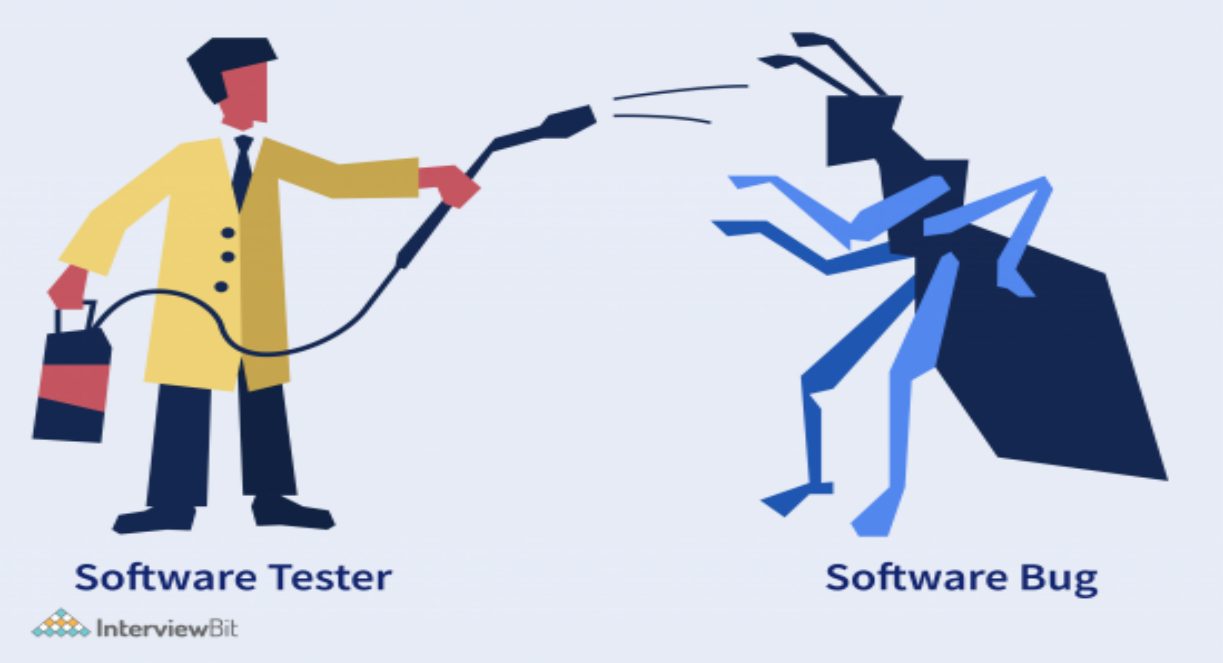


In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered. It could be due to multiple factors, such as the modules might be complicated or the coding related to such modules might be complex.

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. Thus, we prioritize testing on 20% of modules where we experience 80% of bugs.

For an effective testing strategy, it is necessary to thoroughly examine these areas of the software. The defect clustering method relies on the teams' knowledge and experience to identify which modules to test. You can identify such risky modules from your experience. Therefore, the team only has to focus on those "sensitive" areas, saving both time and effort.

5. Pesticide Paradox



In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC). For instance, testing the requirements before coding begins. Amending issues during this stage of a project's life cycle is much cheaper and easier than amending issues at the end of the project when we must write new sections of functionality, resulting in overruns and late deadlines. The cost to fix a bug increases exponentially with time as the development life cycle progresses as shown in the following figure.

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. Therefore, to improve software performance, software testing should begin at the initial phase, that is, during requirement analysis.

6. Testing is Context-Dependent

Each type of software system is tested differently. According to this principle, testing depends on the context of the software developed, and this is entirely true. The reality is that every application has its own unique set of requirements, so we can't put testing in a box. Of course, every application goes through a defined testing process, however, the testing approach may vary based on the application type.

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

The software which we built not only must be 99% bug-free software but also it must fulfill the business, as well as user requirements otherwise it will become unusable software. Even bug-free software may still be unusable if incorrect requirements are incorporated into the software, or if the software fails to meet the business needs.

If you build it, they will come!!! There is a myth that if you build a bug-free system, users will come and use it, but this is not true. 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. So, irrespective of how flawless or error-free a system may be, if it lacks usability and is hard to use, or if it does not match business/user needs, it is only a failure.

Software testing is an incredibly imaginative and intellectual activity for testers. Every software tester should review and understand these 7 principles, as this will help them achieve high-quality standards, as well as give their clients confidence that their software is production-ready. Living by these principles will help your project progress seamlessly. Check them out:

1. Testing Shows the Presence of Defects

As stated in this testing principle, “Testing talks about the presence of defects and doesn’t talk about the absence of defects”. In software testing, we look for bugs to be fixed before we deploy systems to live environments – this gives us confidence that our systems will work correctly

when goes live to users. Despite this, the testing process does not guarantee that the software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.

Even if testers cannot find defects after repeating regression testing, it does not mean the software is 100 % bug-free. For instance, an application may appear to be error-free after passing various stages of testing, but when it is deployed in the environment, an unexpected defect can be found. Team members should always adhere to this concept, and effort should be made to manage client expectations.

2. Exhaustive Testing is Impossible

Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions. No matter how hard you try, testing EVERYTHING is pretty much impossible. The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

Consider the case when we have to test an input field that accepts percentages between 50 and 55, so we test the field using 50, 51, 52, 53, 54, 55. Assuming that the same input field accepts values from 50 to 100, we will need to test using 50, 51, 52, 53, …., 99, 100. This is a basic example. You may think that an automation tool would be able to accomplish this. But imagine a field that accepts a billion values. Will it be possible to test all possible values?

As long as we continue to test all possible scenarios, the software execution time and cost will increase. In order to avoid doing exhaustive testing, we will take into consideration some important testing criteria effects such as risks and priorities as part of our testing efforts and estimates.

3. Early Testing

In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC). For instance, testing the requirements before coding begins. Amending issues during this stage of a project's life cycle is much cheaper and easier than amending issues at the end of the project when we must write new sections of functionality, resulting in overruns and late deadlines. The cost to fix a bug increases exponentially with time as the development life cycle progresses as shown in the following figure.

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. Therefore, to improve software performance, software testing should begin at the initial phase, that is, during requirement analysis.

4. Defect Clustering

In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered. It could be due to multiple factors, such as the modules might be complicated or the coding related to such modules might be complex.

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. Thus, we prioritize testing on 20% of modules where we experience 80% of bugs.

For an effective testing strategy, it is necessary to thoroughly examine these areas of the software. The defect clustering method relies on the teams' knowledge and experience to identify which modules to test. You can identify such risky modules from your experience. Therefore, the team only has to focus on those "sensitive" areas, saving both time and effort.

5. Pesticide Paradox

In software testing, the Pesticide Paradox generally refers to 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. This is based on the theory that when you repeatedly spray the same pesticide on crops in order

to eradicate insects, the insects eventually develop an immunity, making the pesticide ineffective. The same is true for software testing.

Therefore, in order to overcome the Pesticide Paradox, it is imperative to regularly review and update the test cases so that more defects can be found. However, if this process is not followed, and the same tests are repeated over and over again, then eventually there will be no new bugs found, but it doesn't mean the system is 100 % bug free. To make testing more effective, testers must constantly look for ways to improve the existing test methods. To test new features of the software or system, new tests must be developed.

6. Testing is Context-Dependent

Each type of software system is tested differently. According to this principle, testing depends on the context of the software developed, and this is entirely true. The reality is that every application has its own unique set of requirements, so we can't put testing in a box. Of course, every application goes through a defined testing process, however, the testing approach may vary based on the application type.

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

The software which we built not only must be 99% bug-free software but also it must fulfill the business, as well as user requirements otherwise it will become unusable software. Even bug-free software may still be unusable if incorrect requirements are incorporated into the software, or if the software fails to meet the business needs.

If you build it, they will come!!! There is a myth that if you build a bug-free system, users will come and use it, but this is not true. 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. So, irrespective of how flawless or error-free a system may be, if it lacks usability and is hard to use, or if it does not match business/user needs, it is only a failure.

Conclusion

As you have seen, the seven principles of software testing lead to high-quality products. Incorporating these thoughtful principles into your testing can help you gain greater efficiency and focus, as well as improve your overall testing strategy. Added to that, 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.

With this, we conclude our "Principles of Software Testing" blog. Hope you enjoyed reading this article and got a good understanding of what the different principles are.