**Seven Testing Principles**

**Principle 1 - Testing shows presence of defects but cannot prove that there are no defects Principle 2 - Exhaustive testing is impossible Principle 3 - Early testing Principle 4 - Defect Clustering Principle 5 - Pesticide Paradox Principle 6 - Testing is context dependent Principle 7 - Absence of Errors fallacy**

Testing Principles Although testing is itself an expensive activity, the cost of not testing is potentially much higher. The most damaging errors are those which are not discovered during the testing process and therefore remain when the system goes live. Testing shows presence of defects but cannot prove that there are no defects Exhaustive testing is impossible. Risk analysis & prioritisation are used to focus testing effort. Early testing : Testing activity is started as early as possible in the SDLC to find defects early Defect Clustering : Testing effort should be focused proportionally to the expected or later observed defect density of modules. A small number of modules usually contains most of the defects. Pesticide Paradox : If same set of tests are repeated, then NO defects found. To overcome this problem, test cases need to be reviewed and revised regularly to assess different parts of the software or systems. Testing is context dependent : Safety Critical Software is tested differently compared to an e-commerce application Absence of Errors fallacy: Finding & fixing of errors does not help if the system built is unusable and does not fulfil the users’ needs and expectations.

**Here are the 7 Principles:**

**1) Exhaustive testing is not possible**

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

And the million dollar question is, how do you determine this risk?

To answer this let's do an exercise

In your opinion, Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time.

So if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle[Defect](https://www.guru99.com/the-unconventional-guide-to-defect-management.html)Clustering

**2) Defect Clustering**

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems

If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

**3) 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. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let's see this video of the public launch of Windows 98

You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

**4) Testing shows a presence of defects**

Hence, testing principle states that - Testing talks about the presence of defects and don’t talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

This leads us to our next principle, which states that- Absence of Error

**5) Absence of Error - fallacy**

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. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.

To solve this problem, the next principle of testing states that Early Testing

**6) Early Testing**

Early Testing - Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined. More on this principle in a later training tutorial.

**7) Testing is context dependent**

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software’s are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

**Summary of the Seven Testing Principles**

|  |  |
| --- | --- |
| Principle 1 | Testing shows presence of defects |
| Principle 2 | Exhaustive testing is impossible |
| Principle 3 | Early Testing |
| Principle 4 | Defect Clustering |
| Principle 5 | Pesticide Paradox |
| Principle 6 | Testing is context dependent |
| Principle 7 | Absence of errors - fallacy |

Let's review the seven fundamental principles of testing which have been observed over the last 40 years. Knowing how to spot these principles, and how to take advantage of them, will make one a better tester. Especially who are new to testing field this will add to their understanding on software testing. The seven testing principles are as follows:

**Principle 1: Testing shows presence of defects:** Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness. In other words, one can never assume that there are no defects or the application is 100 percent bug free even if thorough testing is done.

**Principle 2: Exhaustive testing is impossible:** Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts. For example, if we are testing a text box that accepts numbers between 0 to 100, we would test for boundary values, one less than boundary value, one more than boundary values, few random numbers, middle number, that’s it and assume that if it is working fine for these numbers it will work for other numbers also. We are not testing for each number from 1 to 100.

**Principle 3: Early testing:** To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives. If the testing team is involved right from the beginning of the requirement gathering and analysis phase they have better understanding and insight into the product and moreover the cost of quality will be much less if the defects are found as early as possible rather than later in the development life cycle.

**Principle 4: Defect clustering:** Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures. The Pareto principle of 80:20 works here, that is 80 percent of defects are due to 20 percent of code! This information could prove to be very helpful while testing, if we find one defect in a particular module/area there is pretty high chance of getting many more there itself.

**Principle 5: Pesticide paradox:** If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, test cases need to be regularly reviewed and revised, and the new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.

**Principle 6: Testing is context dependent:** Testing is done differently in different contexts. For example, safety – critical software is tested differently from an e-commerce site. Very true, testing effort should be based on what is to be tested. Testing focus will depend on what is more important for that type of application.

**Principle 7: Absence–of–errors fallacy:** If the system built is unusable and does not fulfil the user’s needs and expectations then finding and fixing defects does not help. As said, if the product does not meet user’s requirements – explicitly mentioned and implicitly implied, that is if it is not fit for use, there is no point in testing, finding defects and fixing it.

Testing of software is exceptionally imaginative and an intellectual task for testers to perform. Testing of software or applications pursue some principles that are mentioned in this chapter. These principles also play a significant role for a software tester to test the project. These principles are:

1. **Software testing can help in detecting bugs**: Testing any software or project can help in revealing a few or some defects that may or may not be detected by developers. However, testing of software alone cannot confirm that your developed project or software is error free. Hence, it's essential to devise test cases and find out as many defects as possible.
2. **Testing with effectiveness is impossible**: Until your project or application under test has a straightforward structure having limited input, it won't be likely or achievable to check and test all feasible sets of data, modules, and scenarios.
3. **Early testing**: The earlier you will begin to test your project or software the better you will find to utilize your existing time.
4. **Defect in clustering**: At the time of testing, you can observe that majority of the defects or bugs reported are because of a small number of modules inside your software or system.
5. **Software testing is context-dependent**: Various methods, procedures, and kinds of testing are there which defines the type and characteristics of the application. For example, an application related to health device needs more testing and doctor based feedbacks than a game or small software.
6. **Errorfree or Bug-free software is a myth**: Just because when a tester tested an application and didn't detect any defects in that project; doesn't indicate or imply that your software is ready for shipping.

So, at the time of testing modules or working of software, you as a tester need to test whether your software is meeting all the requirements of the user or client or not and whether the bugs found during testing has been mended or not. These many factors need to be considered before shipping the software or releasing it to the market.

**Principles of Testing**– There are seven principles of software [**testing**](http://tryqa.com/what-is-a-software-testing/). They are as follows:

**1) Testing shows presence of defects:** Testing can show the [**defects**](http://tryqa.com/what-is-defect-or-bugs-or-faults-in-software-testing/) are present, but cannot prove that there are no defects. Even after testing the application or product thoroughly we cannot say that the product is 100% defect free. Testing always reduces the number of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

**2) Exhaustive testing is impossible:** Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing [**we can use risks**](http://tryqa.com/what-is-risk-in-software-testing/)and [**priorities**](http://tryqa.com/what-is-the-difference-between-severity-and-priority/) to focus testing efforts. For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30  517  578  125  (515) tests. This is very unlikely that the project timescales would allow for this number of tests. So, accessing and managing risk is one of the most important activities and reason for testing in any project.

**3) Early testing:** In the [**software development life cycle**](http://tryqa.com/what-are-the-software-development-life-cycle-phases/) testing activities should start as early as possible and should be focused on defined objectives.

**4) Defect clustering:** A small number of modules contains most of the defects discovered during [**pre-release testing**](http://tryqa.com/what-is-alpha-testing/) or shows the most operational failures.

**5) Pesticide paradox:** If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, it is really very important to review the test cases regularly and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

**6) Testing is context dependent:** Testing is basically context dependent. Different kinds of sites are tested differently. For example, safety – critical software is tested differently from an e-commerce site.

**7) Absence – of – errors fallacy:** If the system built is unusable and does not fulfil the user’s needs and expectations then finding and fixing defects does not help.

# Seven Principles of Software Testing

Updated: December 2, 2018 - [Amir Ghahrai](https://www.testingexcellence.com/author/amir-ghahrai/)

Software testing is an extremely creative and intellectually challenging task. When testing follows the principles given below, the creative element of test design and execution rivals any of the preceding software development steps.

## 1. Testing shows the presence of bugs

Testing an application can only reveal that one or more defects exist in the application, however, testing alone cannot prove that the application is error free. Therefore, it is important to design test cases which find as many defects as possible.

## 2. Exhaustive testing is impossible

Unless the application under test (AUT) has a very simple logical structure and limited input, it is not possible to test all possible combinations of data and scenarios. For this reason, risk and priorities are used to concentrate on the most important aspects to test.

## 3. Early testing

The sooner we start the testing activities the better we can utilize the available time. As soon as the initial products, such the requirement or design documents are available, we can start testing. It is common for the testing phase to get squeezed at the end of the development lifecycle, i.e. when development has finished, so by starting testing early, we can prepare testing for each level of the development lifecycle.

Another important point about early testing is that when defects are found earlier in the lifecycle, they are much easier and cheaper to fix. It is much cheaper to change an incorrect requirement than having to change a functionality in a large system that is not working as requested or as designed!

## 4. Defect clustering

During testing, it can be observed that most of the reported defects are related to small number of modules within a system. i.e. small number of modules contain most of the defects in the system. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

## 5. The pesticide paradox

If you keep running the same set of tests over and over again, chances are no more new defects will be discovered by those test cases. Because as the system evolves, many of the previously reported defects will have been fixed and the old test cases do not apply anymore. Anytime a fault is fixed or a new functionality added, we need to do regression testing to make sure the new changed software has not broken any other part of the software. However, those regression test cases also need to change to reflect the changes made in the software to be applicable and hopefully fine new defects.

## 6. Testing is context dependent

Different methodologies, techniques and types of testing is related to the type and nature of the application. For example, a software application in a medical device needs more testing than a games software. More importantly a medical device software requires risk based testing, be compliant with medical industry regulators and possibly specific test design techniques. By the same token, a very popular website, needs to go through rigorous performance testing as well as functionality testing to make sure the performance is not affected by the load on the servers.

## 7. Absence of errors fallacy

Just because testing didn’t find any defects in the software, it doesn’t mean that the software is ready to be shipped. Were the executed tests really designed to catch the most defects? or where they designed to see if the software matched the user’s requirements? There are many other factors to be considered before making a decision to ship the software.

Other principles to note are:

* + **Testing must be done by an independent party.**

Testing should not be performed by the person or team that developed the software since they tend to defend the correctness of the program.

* + **Assign best personnel to the task.**

Because testing requires high creativity and responsibility only the best personnel must be assigned to design, implement, and analyze test cases, test data and test results.

* + **Test for invalid and unexpected input conditions as well as valid conditions.**

The program should generate correct messages when an invalid test is encountered and should generate correct results when the test is valid.

* + **Keep software static during test.**

The program must not be modified during the implementation of the set of designed test cases.

* + **Provide expected test results if possible.**

A necessary part of test documentation is the specification of expected results, even if providing such results is impractical.