[Print](javascript:window.print())

**Course Transcript**

Fundamentals of Software Testing

**Introduction to Software Testing**

[1. The Necessity of Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t2)

[2. What is Software Testing?](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t5)

[3. Meeting Software Test Objectives](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t8)

[4. General Software Testing Principles](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t11)

[5. Applied Software Testing Principles](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t32)

[6. The Importance of Software Testing and Mitigating Harm](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t14)

**Process Activity and the Psychology of Testing**

[1. The Fundamental Software Test Process](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t17)

[2. Exit Criteria and Test Closure Activities](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t20)

[3. The Psychology of Software Testing](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t23)

[4. Contrasting Software Testers and Developers](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t36)

[5. Organizing Testers and the Software Test Process](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#t26)

The Necessity of Software Testing

Learning Objectives

*After completing this topic, you should be able to*

* *recognize different types of software errors*
* *recognize how testing can improve quality in software development*

**1. Correcting Errors, Defects, and Failures**

Software systems form an integral part of our daily lives. Whether in our homes or at our workplaces, software systems have become indispensable for a comfortable existence.

Therefore, software failure can prove expensive and result in the loss of time, effort, and reputation.   
  
For critical software, these failures can also cause major financial losses, injury, or result in the loss of life.

Let's consider a software program used to issue airplane tickets online. Suppose the program used to create reservations miscalculates airfare because it erroneously inserts an extra zero in a numeric field used to perform the airfare calculation. In such a situation, if the airfare calculated was uncompetitively high, potential customers would likely refuse to pay the price and take their business elsewhere. This could cause a loss of reputation for the airline along with the lost business.

In another instance, suppose a refrigerator is rolled out with an incorrect automatic defrost temperature. In this case the refrigerator might overheat resulting in complete failure of the appliance. This failure may require the company to arrange for a recall or a complete replacement of the appliance for all new customers. This, in turn, would result in a loss of and a negative image for the brand.

In more serious cases, faulty software systems may result in injury or loss of life.  Consider a software system that miscalculates the quantity of chlorine needed to disinfect the water in a reservoir. Suppose a decimal point is wrongly placed so that the amount of required chlorine is double the permissible amount. This could be fatal for the people using the water.

In another instance, a software error in a medical machine could have serious implications for any patient using the machine.

This is why it is important to locate and rectify the defects of a software system before it is released for public use.

To eliminate the defects in faulty software, it is important to understand what causes the defects. Faulty software is a result of the errors (or mistakes) made while designing and building it.  
  
Most software issues can be divided into three categories. These categories are

**error**

An error (or mistake) is an action performed by a person or persons that leads to an incorrect result.  This could be an error made by someone using a software system, or an error made by someone in the process of designing and building the software system.

**defect**

A defect, also known as a bug or fault, is a flaw in a component or a system that can cause the component or system to fail to perform its required function such as an incorrect statement or data definition.

**failure**

A failure is defined as a deviation of the component or system from its expected delivery, service, or result.  
  
A defect causes a software system to fail. Failures occur when a software system doesn't perform as expected by the end user, or executes an action which it shouldn't have. It is important to note that not all defects result in a failure – a failure only occurs when code is executed.

Errors tend to increase due to stringent timelines, inexperience, careless working habits, complex work situations, or even misinformation.  
  
Failures may also be caused by environmental factors including, but not limited to, climate change, pollution, and magnetic fields.

Though not all errors translate into failures, errors can lead to a defect, which can, in turn, cause a failure. You can reduce the likelihood of an error occurring in the future by conducting a root cause analysis.

Root cause analysis involves trying to track the failure of a system all the way back to its root cause.  Understanding and eliminating the root cause of a failure results in software process improvement, which should prevent the recurrence of related failures in the future. This should bring about an overall improvement in the quality of those systems.

Root cause analysis is an important part of quality assurance. Root cause analysis can help to anticipate defects in a product, and tackle them at an early stage.

Defects can arise in any of the four stages in a product life cycle – analysis, design, development, and implementation.  
  
Based on these stages, here are four scenarios to consider.

Graphic

*The table is composed of four rows and four columns. The four columns are named Analysis, Design, Development, and Implementation. The rows represent requirements one, two, three and four. All four cells of the first row represent a zero-defect delivery scenario. A box at the end of the first row says that in such a scenario, all correct attributes of the product are delivered.  
In the second row the first two cells represent zero defects while the third and fourth cells represent defects. A box at the end of the second row says that in such a scenario, defects can be corrected.  
In the third row only the first cell represents zero defect while the second, third and fourth cells represent defects. A box at the end of the third row says that in such a scenario, redesign is required to correct defects.  
In the fourth all the cells represent defects. A box at the end of the fourth row says that in such a scenario, correction may be impossible.*

**requirement 1**

Requirement 1 is an ideal scenario where you have been able to completely comprehend the customer needs. The designer of the product has succeeded in understanding customer needs and designs a product to meet those needs.  
  
The product is built in accordance with the design and as a result, the product works as expected and houses all attributes required for flawless execution.

**requirement 2**

In the scenario for requirement 2, all works well until the product is designed. Defects occur while the product is being built or coded. Such defects are ‘internal’ and can be detected during testing. Such flaws can be easily corrected.

**requirement 3**

In requirement 3, a defect is introduced during product design. Such defects are difficult to correct. To eliminate such a flaw, it becomes imperative to map customer requirements to the design. If the mapping is not conducted, such errors will go undetected during the testing phase and when detected later, they will be difficult to fix as it would require considerable redesigning.

**requirement 4**

Requirement 4 is the most difficult to address. This is caused when customer needs have not been comprehended, or have not been communicated effectively. You might design a flawless product from your understanding of the requirement. The product might also pass all tests but will be rejected because it does not cater to the customer’s expectations. Such defects appear only during the test or live phases and result in considerable financial loss.

Other than understanding the impact of defects at various phases of a product life cycle, it is also important to understand the effects of fixing defects on cost and time.

Graphic

*The connection between time and cost is represented by a graph. The X axis of the graph represents time while the Y axis represents cost. There are five boxes aligning the X axis – Analysis, Design, Development, Test, and Use.*

If you misunderstand a client requirement, and detect the defect during the needs analysis stage, it becomes very easy to correct. A new requirements document could be issued after a discussion with the customer. Again, if the same defect is detected during the design phase of the product, the requirements are mapped to the design and easily corrected with minimal expenditure.

Graphic

*The passage of time is represented by a dotted line appearing from the beginning of the axes and ending on the block labeled Design. The line is a curved upward line.*

But if such a defect is detected after the product is launched, the cost to fix the defect could be huge, as it demands high level design and code changes. To lower expenses related to testing and defect fixes, it becomes imperative to eliminate any possible error as soon as it occurs. Therefore, the cost of fixing a defect is directly proportional to the passage of time.

Graphic

*The dotted line increases further and ends on the block labeled 'Use'. The words, 'The cost of investigating and correcting defects is directly proportional to the increase in time.' appears.*

Question

Match the definition to the correct software testing term.

**Options:**

1. An action by a user that produces a result from the system that is incorrect.
2. A flaw in a component or system
3. A deviation of a component or system from its expected result

**Targets:**

1. Failure
2. Defect
3. Error

Answer

*A failure is defined as a deviation of the component or system from its expected delivery, service or result.*

*A defect is a flaw in a component or a system that can cause the component or system to fail to perform its required function such as an incorrect statement or data definition.*

*An error is an action by a user that produces a result from the system that is incorrect.*

**Correct answer(s):**

Target 1 = Option C

Target 2 = Option B

Target 3 = Option A

**2. Software testing as a solution**

Quality is how well a component, system or process is designed, and how well it conforms to the design.  
  
Testing helps to improve your confidence that a product meets criteria for quality.  
  
You must make sure that the testing has been thorough. Weak tests may provide a false sense of security because they may not uncover defects.  This results in defects being uncovered only during execution.

Testing is a thorough procedure and tests for both

**functional attributes**

Testing for functional attributes ensures that the product performs an expected task.

**non-functional attributes**

Testing for non-functional attributes measures how well or fast a task is performed. Testing non-functional attributes usually requires a metric, such as time to complete a task.

In addition to testing attributes, part of the testing effort is focused on verification and part on validation. Verification evaluates a product to determine that it meets the requirements set out for it. Validation ensures that it meets the needs of users, and that it is fit for the purpose for which it was built.  
  
It is also important to maintain focus on the variables of time and cost. The best solution is one that delivers as promised, within the allotted time frame while remaining within budget.

You might design and develop a product that fits a client’s specification, but if it significantly exceeds the client’s budget the status of the project as a success could be compromised.  
  
For example, if a customer wants a simple scanning application, designing a complex integrated scanning, photocopying, collating and mailing system for them will add needless complexity, cost, and time to what should have been a simple system.  
  
Similarly, if the testing phase for the scanner continues for so long that it results in missing the targeted release date of the product, it again results in increased cost and customer dissatisfaction.

It is essential to understand the expectations of customers and then create a testing plan. We could sum up customer expectation based on four perceptions of quality that include

**physical features**

When the quality of a product is gauged simply by observing its physical features, the customer might only want the superficial attributes of the product to be quantified.

**usability**

When the customer gauges quality by assessing the product's usability, and not just by quantitative methods, then simply making sure the product can achieve the task is not enough. The product would not be ready for release until the usability meets their criteria.  
  
This could mean performing a task in a given timeframe, or allowing users to use it with minimal training.

**industry standards**

Proper software testing can give a quantitative measure of software quality for both functional and non-functional requirements.

**subjective responses**

Customer reactions to quality can often be a gauge to their expectations and willingness to believe that the product will meet their needs.

The purpose of testing is to eliminate defects and assure a quality product. Complete or exhaustive testing is defined as a test approach in which the test suite comprises all combinations of input values and preconditions. Exhaustive testing, though ideal, is neither practical nor advisable. Which brings us to the question – how much testing is enough?

To counter the problem, you should focus your testing efforts on risk and priorities. Exhaustive testing is impossible in all but a very small number of cases. Take for example a text input field. Even if we only consider alphanumeric input, there are 62 possible inputs for each character space available. The number of possible combinations is enormous – 14 million combinations for just 4 character spaces. Clearly exhaustive testing would be impossible.

You can provide customer satisfaction only when you deliver a product that satisfies the customer’s needs and is delivered on time and does not exceed the budget.

A test approach should be devised by comparing the test with the risks, which the customer, the project stakeholders, the project as an entity, and the software is exposed to.  
  
What should be considered is the level of risk – technical or business – and time and budget constraints.  
  
A risk assessment will decide the amount of testing that is advisable. This amount will vary depending on the level of risk involved. Testing should also supply project stakeholders with requisite information which will help them to make informed decisions about software release for the next stage in development or roll-out to customers.

**3. Summary**

Software systems are an increasingly common part of life, from business applications to consumer products; yet, most people have experienced software that did not work as expected. Defective software could lead to a loss of money, time, reputation, and could even cause injury or death.  
  
A human being can make an error, which results in a defect in the software. When the code is run this can lead to a failure.  
  
Proper testing of software can help to reduce the risk of problems occurring during operation and contribute to the quality of software.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

What is Software Testing?

Learning Objective

*After completing this topic, you should be able to*

* *recognize the different goals in testing*

**1. Testing as a process**

The main objective of testing is to discover defects. Other objectives include preventing defects, and getting a measure of the quality of software, often to increase confidence. Testing as a process consists of activities before, during, and after actual testing occurs.

Testing includes a range of activities that form an integral part of every stage of the software life cycle. You should always try to detect and fix defects during the initial stages of software development as it is a more economical option. If defects are detected during the analysis phase while verifying the requirements of the client, it can save a lot of headaches and potential costs related to finding the defects later in the process.  
  
Testing during the requirement phase involves checking the *test basis* – client requirements and design specifications – against the test design.

Note

*A test basis includes all software related documentation which specifies the requirements of a customer. If a document can be changed only by a formal amendment procedure, then the test basis is referred to as a frozen test basis.*

Various types of testing methods are used during each stage of the software life cycle. You can classify testing into static and dynamic testing. When you conduct tests by running a software program it is known as dynamic testing. Dynamic testing, though less cost-effective than static testing, is a more thorough testing procedure than static testing. Dynamic testing involves executing the code for an application and generally requires more effort in terms of cost and time.

Static testing includes reviewing software related documentation and source code. This type of testing is more economical than dynamic testing and helps deliver an overview of the quality of the software.  
  
The static testing methods would form a part of the activities such as analyzing customer requirements and specifications, creating the design for the software, and writing the instructional manual. Dynamic testing would be required after you have written code for a functional piece of software.

Planning is key to software testing and includes processes that take place before and after test execution. Planning may include activities such as creating a test plan and test strategy. Planning also includes defining entry and exit criteria.

After creating a test plan, preparation is the next step in software testing. Preparation involves determining test conditions and designing *test cases*.

Note

*A test case is a set of input values, execution preconditions, expected results, and execution post conditions. A test case is developed with a defined objective or test condition. This is done to exercise a particular program path or to verify compliance with a specific requirement.*

After test execution, you should evaluate the test to check whether it has satisfied all preconditions stated in the test plan. Also, it is important to check whether enough testing has been conducted and if it is appropriate to end the testing process.

Different testing viewpoints take different objectives into account. Here are three objectives, along with some information in general terms about when they might be considered. These include

**finding defects**

In development testing, which includes component, integration and system testing, the main objective may be to cause as many failures as possible in order to identify defects and fix them as early as possible in the software life cycle.

**gaining confidence in product quality and providing information**

Later in the software life cycle during acceptance testing, the main objective may be to verify that the software works as anticipated, and gain confidence that requirements have been met. A similar objective at this phase may be to provide information about the state of the system to stakeholders in order to assess the quality of the software (with no intention of fixing defects).  This is done primarily to gauge the risk of releasing the system at a given time.

**preventing defects**

In maintenance testing, the main objective may be to ensure that no new defects have been introduced during development of changes.

Other than development and acceptance testing, you also need to conduct maintenance and operational testing.  
  
Maintenance testing is done on an existing and operational system. It might be required after modifications to the system, or migration.  
  
The main objective of operational testing is to check for characteristics such as reliability and availability. Attributes such as reliability and availability surface during the operational use of the software. Reliability is a set of attributes representing the capability of the software to maintain a level of performance under stated conditions for a stated period of time. Availability refers to whether the product is ready for use whenever required.

Testing is an iterative process, and organizations will improve their testing capability and knowledge with time. This will have a positive knock-on effect for future projects in terms of quality and assurance.

Question

What stages of a software life cycle would require dynamic testing?

**Options:**

1. Analyzing customer requirements and specifications
2. Creating the detailed design for the software
3. Building functional software

Answer

***Option 1:****Incorrect. Analyzing customer requirements would involve static testing as it would only include studying requirement related documentation.*

***Option 2:****Incorrect. Creating detailed design involves static testing. Here you only check the design plan against the requirements stated by the customer.*

***Option 3:****Correct. Dynamic testing involves executing the code, and is required to test functional software after it has been developed.*

**Correct answer(s):**

3. Building functional software

An effective software testing plan should include both testing as well as debugging the software.   
Debugging is performed as a result of testing.  
  
Debugging is performed by developers to uncover where a defect in the code exists and correct it. Debugging can be performed on code or on requirements and specifications. Debugging might not study the effects of a correction on the other components of a system.

Testing studies various components of a system with the aim to discover defects and report them; testing doesn't involve fixing the defects. As opposed to debugging, testing also checks the effects of corrections on the other components of a system.

A detailed debugging exercise is important to make the software worthy enough to complete the testing phase. It is a waste of time for testers to uncover errors which could have been easily corrected during debugging.  
  
While debugging ensures partial confidence in a product, testing attempts to rigorously study every component of a system and reports defects. Testing also repeats tests to check whether the corrections made by developers are actually effective.

**2. Summary**

Testing includes a range of activities that form an integral part of every stage of a software life cycle. Testing activities may be both dynamic as well as static. Both dynamic and static methods ensure an effective testing plan.  
  
Testing has a number of objectives, Depending on your point of view, the main objective could be finding defects, gaining confidence about the level of quality and providing information, or preventing defects.  
  
Debugging is an important activity. While testing detects failures that are caused by defects, debugging involves identifying the cause of a defect, repairing the code, and checking that the defect has been fixed correctly.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

Meeting Software Test Objectives

Learning Objective

*After completing this topic, you should be able to*

* *recognize how software testing meets different test objectives*

**1. The Objectives of Software Testing**

The main objective of software testing is to uncover defects and resolve them. Testing can also be conducted to provide information about the software to its stakeholders and increase confidence levels by ensuring the quality of the product.  
  
You can perform tests to check certain attributes of the software. For example, you can check for an attribute like Defect Density by dividing the number of design defects found in down-stream activities by a measure of product size, such as function points or physical source lines of code. This type of testing will help you measure product reliability and assess the risk of releasing the software.

Other than ensuring quality, you should also consider the point of view of end users – customers who purchase the software to use it to complete their daily tasks. These customers wouldn't be interested in the number of defects in the software unless they are affected by them.  
  
For example, a word processor might have a few lines of erroneous code, but as long as a user can use it to type documents and format them, the software would qualify as fit for use.

Even if an end user qualifies a product as usable, it does not necessary mean that the product is free from defects. Thus, in spite of the end user perspective, your attempt should be to free software of critical defects.  
  
While reviewing defects and failures, you may arrive at the root cause of a defect. A root cause analysis should uncover the real reason why an error occurs and help you improve testing processes. The analysis also aids in refining the quality of your test conditions and the general software development process.  
  
Users might be logging a number of bugs about a feature which is working properly, but is complex to operate. In such a scenario, you could design a step by step wizard which teaches an individual to use the function. This wizard could be invoked by starting to use the function in question. This in turn, improves the design and subsequent development of the product.

Defects tend to gather in single feature or set of features and form a cluster. This might happen because a certain area of the source code is relatively complicated. These areas are known as hot spots, and as a tester you should concentrate on hot spots during risk assessment.

Clusters may also be created when a newly introduced correction results in a chain of defects on various parts of the software. These defects are referred to as knock-on defects.

Early review cycles and static tests always prove to be a more economical option and help you identify potential defect clusters. So you should begin testing as early as possible in the software development cycle.

Defect clusters have a tendency to change over time, so if the same set of tests is conducted repeatedly, they will fail to discover new defects. This can be referred to as the pesticide paradox.   
  
To eliminate the effects of the pesticide paradox, you should continually review existing tests and evolve newer ones to check varied parts of a component and uncover new defects.

The pesticide paradox can be illustrated with an animation of a pesticide can. When a test plan is created for the first time, it is successful in uncovering relevant defects and correcting them. But as defect clusters tend to change with time, the same set of tests would be redundant for correcting defects.

Animation

*The animation shows a can of pesticide being sprayed over a group of bugs which get killed as a result, but the spraying continues in the same area and fails to kill the other surrounding bugs.*

The pesticide paradox illustrates that an effective testing approach should always integrate new tests. By creating effective static testing techniques and improving the software development cycle, you'll encounter fewer defects during a dynamic test.  
  
An improved testing initiative will uncover a greater number of defects.   
  
Initial prevention during the Analysis phase reduces the number of defects that may appear while operating the software. Later prevention during the Development phase ensures that you create software with very few defects.

Graphic

*The image represents a bar graph which shows the reduction of defects as a result of testing over a number of software releases. The Y axis represents the number of defects and has a range of 0 to 100. The axis is marked at regular intervals of 0, 20, 40, 60, 80, and 100. The X axis represents the number of releases and has a range of one to 10. The axis is marked at regular intervals of 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. The progress of defect prevention is shown by four different groups of bars. These bar groups can be classified to include the total number of defects, represent the total number of defects, defects identified during reviews, defects identified during testing, and failures which occur during live use. All these bar groups are shown as decreasing steadily in size from release 1 to release 10.*

As defect clusters are eliminated, your goal should be to change the focus of your testing efforts. Apart from uncovering defects, you should also strive to bring about a general improvement in software design documents and development processes.  
  
This in turn will help you lower the costs associated with the testing initiative.

In an effort to bring about an improvement in software quality, you should also include debugging as an important activity. Debugging is the process of finding, analyzing, and removing the causes of failures in software.   
  
Debugging and testing are varied activities:

**Debugging**

Debugging is conducted by programmers. When a tester discovers a defect in the code, a programmer locates the defect and its immediate cause. The programmer then corrects the code and runs it.  
  
The tester verifies whether the defect has been fixed in the manner expected. In certain cases, the programmer might test their fixes themselves. However, debugging solves only a single defect in the code, and does not analyze the effect of a fix on other parts of the code.

**Testing**

As opposed to debugging, testing shows that defects are present.  
  
Testers uncover defects but programmers solve them. Although testing reduces the probability of undiscovered defects remaining in the software it cannot prove that software is defect free.

Question

You've created a test plan for a software application. How would you ensure that the tests find and prevent defects if any are present?

**Options:**

1. Create a detailed debugging plan to precede testing
2. Rely only on testing efforts for uncovering defects
3. Standardize a set of tests for every release of the software
4. Continue to develop and add new tests to the plan
5. Incorporate a healthy mix of static and dynamic tests

Answer

***Option 1:****Incorrect. Debugging is performed as a result of testing. Debugging is not a testing activity, but a development activity.*

***Option 2:****Incorrect. It is important to debug software before it actually gets into the testing cycle. This is because programmers detect quite a few obvious defects in the code and fix them immediately. Having such defects remaining in the software would only lead to a waste of time and increase testing related costs.*

***Option 3:****Incorrect. Standardizing tests across releases lowers the chances of finding new defects. You might have created a group of tests to discover and prevent a particular kind of defect. After such defects are found and resolved, these tests may become redundant for the rest of the software and may result in the pesticide paradox.*

***Option 4:****Correct. It is a good idea to constantly develop and evolve the testing plan. What may have been applicable for a set of defects in Release 1 of the software might not be relevant in Release 4 where there is a possibility of new defects being introduced into the code as a result of changes made previously.*

***Option 5:****Correct. A healthy balance of static and dynamic testing should be incorporated in an effective testing plan. Dynamic testing, though less cost-effective than static testing, is a more thorough testing procedure than static testing. Whereas, static testing includes reviewing documentation and source code.*

**Correct answer(s):**

4. Continue to develop and add new tests to the plan  
5. Incorporate a healthy mix of static and dynamic tests

Software testing has three clearly defined objectives:

* uncovering defects
* gaining confidence about the level of quality and providing information
* preventing defects

Uncovering defects is the main goal of initial testing. In the early stages of the product development cycle, you can conduct static tests to review specification documents and the initial design of the software. For example, if you're developing accounting software, you can map the initial design to the specifications stated by the customer. This is done to check whether your team has been able to understand client needs.

After the design has been reviewed, and development begins, you can execute the code to uncover defects. These tests form part of *development testing* and include *component* and *integration testing*. The objective is to cause as many failures as possible so that a maximum number of defects are uncovered and solved.

The objective of testing may not always be to uncover defects but to gain confidence in the product and provide information about its quality.  
  
*Acceptance testing* attempts to establish confidence in the system. *User acceptance testing* typically verifies that the system is fit for use by users. *Operational acceptance testing* confirms that the system is acceptable to system administrators. *Contract* and *regulation acceptance testing* is performed against contractual or regulatory criteria. *Alpha* and *beta testing* garners feedback from potential or existing customers. *Alpha testing* is typically performed at the developer’s site, whereas *beta testing* is typically done “in the field” at customers’ sites.

The last objective of testing is to uncover new defects which may have been introduced into the software from previous fixes. This testing can be referred to as *regression testing*. During regression testing, you can ensure that a change made to fix a defect has not affected any other part of the software that it should not have affected.

Question

You have completed your first testing cycle for the accounting software. It is important for you to test whether there have been any unwanted changes in the software as a result of a defect fix. What kind of testing would satisfy your testing objective?

**Options:**

1. Development testing
2. Acceptance testing
3. Regression testing
4. Component testing

Answer

***Option 1:****Incorrect. Development testing occurs after a product has been developed and its main aim is to create as many failures as possible to uncover and fix defects. Development testing forms a part of initial testing activities and happens right after the design stage. Regression testing would help you to check whether any unwanted changes have occurred due to a fix.*

***Option 2:****Incorrect. Acceptance testing checks whether a product is fit for use and whether the end user will be satisfied with the product. This testing does not eliminate defects but provides information about the quality of the product and ensures whether the product can be released for use. You would need to use regression testing to test for any unwanted changes in the software.*

***Option 3:****Correct. Regression testing ensures that no errors have been introduced while changing software. Software might have been changed to fix a defect and that change might have had an adverse effect on some other part of the software. Such effects are undesirable and should be prevented through regression testing.*

***Option 4:****Incorrect. Component and integration testing form a part of development testing. Development testing forms a part of initial testing activities and happens right after the design stage. Regression testing would help you to check whether unwanted changes have occurred due to a fix.*

**Correct answer(s):**

3. Regression testing

**2. Summary**

Testing has three objectives: finding defects, gaining confidence about the level of quality and providing information, and preventing defects.  
  
Different viewpoints in testing take different objectives into account. For example, in development testing the main objective may be to cause as many failures as possible so that defects in the software are identified and can be fixed. In acceptance testing, the main objective may be to confirm that the system works as expected. Regression testing checks that no new defects have been introduced during development of changes.   
  
Debugging and testing are different. Testing can show failures that are caused by defects.  
Debugging is the development activity that identifies the cause of a defect, repairs the code, and checks that the defect has been fixed correctly.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

General Software Testing Principles

Learning Objective

*After completing this topic, you should be able to*

* *recognize the fundamental principles in testing*

**1. General Principles of Software Testing**

Software testing is based on certain principles that have evolved over the years. These principles guide testers and reduce software related problems.

There are seven principles of software testing. Some of these principles are referred to as general software testing principles, whereas the others are referred to as applied software testing principles.

General software testing principles provide a standard framework to testers for conducting tests and discovering defects including

* testing shows presence of defects – Principle One
* exhaustive testing is impossible – Principle Two
* confusing an absence of errors with product fit is a fallacy – Principle Three

The first principle states that running tests on software will discover defects, but that the absence of defects does not mean that the software is defect free. These tests also decrease the possibility that the software still contains defects, but testing an application doesn't mean that you will be able remove all defects from the software.  
  
For example, you might compile a list of tests for testing an online shopping site. You might not be able to discover any functional defects. When the site goes live, however, and is used by a large number of users in varying ways, errors and defects may then come to light.

The second principle states that it's impossible to conduct exhaustive testing. Exhaustive testing can be defined as a test approach in which all possible data inputs and preconditions are used. Testing every data input and precondition of software is not feasible because it results in an increase in the cost of testing and also overshoots the time allotted to test the software.  
  
Exhaustive testing is possible only when the application under test (AUT) is based on simple logic and requires minimal user input. But when the application uses complex logic and requires a lot of user input, it is impossible to conduct an exhaustive test on the application. So instead of testing every parameter, it becomes important to base your testing efforts on risks and priorities.

Consider the example of a web page that prompts users to input information for a children's dance competition. The web page has only a single input field. The field has been designed to enter the age of a child. The permissible age range for entering the competition is between five and 12 years, so the only criteria you need to test here is whether the age entered falls in the range of five to 12. In such a scenario, you would be easily able to conduct exhaustive testing.

But consider the example of web page used to record user’s delivery addresses and parse them against a zip code database. These fields would have criteria based on numerous permutations and combinations. In such an instance, exhaustive testing would be impossible.

The third principle outlines the absence-of-errors fallacy. The fallacy states that although testing might succeed in locating and correcting all possible defects in the software, the software itself might not be fit for use by an end user. This might happen as a result of misinterpreting customer expectations, or as a result of poor usability.

For example, you design a word processor that formats, spell checks, and prints documents. The software could pass testing and perform all these functions, but if the customer required language translation and thesaurus functionality then the software does not meet the customer’s needs. The absence of errors in testing does not mean that the software is suitable for this customer.

Question

You have designed an application that calculates wind resistance for concept car 3D models. You have tested the application and it has passed testing. A salesperson selling the product tells a client “our application has passed our rigorous testing, and so is 100% error free!” Which principle of software testing is your salesperson not familiar with?

**Options:**

1. Testing shows presence of defects – Principle One
2. Exhaustive testing is impossible – Principle Two
3. Confusing an absence of errors with product fit is a fallacy – Principle Three

Answer

***Option 1:****Correct. The first principle states that, although testing shows the presence of defects and can also reduce the probability of discovered defects, it can never be proof that the software is defect free.*

***Option 2:****Incorrect. While exhaustive testing may have bolstered the salesperson’s claim, principle two says that such testing is (in practical terms) impossible.*

***Option 3:****Incorrect. The statement by the salesperson says nothing about product fit.*

**Correct answer(s):**

1. Testing shows presence of defects – Principle One

Question

You need to design tests for a very basic authentication system used by an airline ticketing system that takes a 10 digit number and validates if it is one of three pre-defined values. You realize that you would need 9,999,999,999 test cases to check every possible combination, and even then it would exclude cases where the user pressed **Enter** accidentally, used **Backspace**, or even had a long pause between numbers. Which software testing principle would use to design tests for this software?

**Options:**

1. Testing shows presence of defects – Principle One
2. Exhaustive testing is impossible – Principle Two
3. Confusing an absence of errors with product fit is a fallacy – Principle Three

Answer

***Option 1:****Incorrect. The first principle states that running tests on software prove that defects are present. The test will also decrease the possibility of defects being left over in the software, but testing cannot prove that the software is defect free.*

***Option 2:****Correct. The second principle states that it is not feasible to conduct exhaustive tests on software. Exhaustive testing is a test approach that tests all possible permutations of input. The principle that exhaustive testing is an unfeasible practice is more applicable in this scenario.*

***Option 3:****Incorrect. The third principle is based on the fact that it is a fallacy to assume that, although no defects are present in the software, it is fit for use. It may happen that a customer requirement is misunderstood resulting in a faulty product, as a result, the application fails to perform the task expected by an end user. You would use principle two to design tests for this software.*

**Correct answer(s):**

2. Exhaustive testing is impossible – Principle Two

Question

You have designed tests for a system designed to help workers in two geographically distant offices collaborate over the Internet. Unfortunately, all traffic for these offices must travel through a commercial proxy system, and this was not identified by your team during the requirements phase. When you tested the system, it ended up working perfectly in the test environment, but had no support for the office proxy server. Which software testing principle did your team overlook while designing tests?

**Options:**

1. Testing shows presence of defects – Principle One
2. Exhaustive testing is impossible – Principle Two
3. Confusing an absence of errors with product fit is a fallacy – Principle Three

Answer

***Option 1:****Incorrect. The first principle states that running tests on software will indicate the presence of defects if there are any. The test will also decrease the possibility of defects being left over in the software, but testing is not proof that the software is completely defect free.*

***Option 2:****Incorrect. The second principle states that it is impossible to conduct exhaustive testing. Exhaustive testing can be defined as a test approach in which all possible data inputs and preconditions are used. The team overlooked the principle that an absence of errors does not necessarily mean that the software is fit for use.*

***Option 3:****Correct. The third principle is based on the fact that, although testing may locate and solve many defects, it may still be unfit for release. A customer requirement may have been misunderstood resulting in a faulty product design and, as a result, the application fails to perform the task expected by an end user. The team did not pay attention to the principle that an error free software is not necessary a usable one.*

**Correct answer(s):**

3. Confusing an absence of errors with product fit is a fallacy – Principle Three

**2. Summary**

There are three general principles that guide testers while devising software testing plans.  
  
The first principle states that, although testing shows the presence of defects and can also reduce the probability of discovered defects, it can never be proof that the software is defect free.  
  
The second principle discusses that exhaustive testing is both impossible and unfeasible in context to time and cost, and is only possible with instances of an extremely simple logical structure and limited input.  
  
The third principle outlines the absence-of-errors fallacy. The fallacy states that although testing might succeed in locating and correcting all possible defects in the software, the software itself might not be fit for use by an end user. This might happen as a result of misinterpreting customer expectations.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

Applied Software Testing Principles

Learning Objective

*After completing this topic, you should be able to*

* *recognize the applied software testing principles*

**1. Applied Principles of Software Testing**

You should always base your software testing efforts on certain established principles. Other than the three general principles previously discussed, software testing also follows four applied principles. Applied testing provides a standardized format for creating test plans, and acts as a guide to effective testing.

The applied principles of software testing concern the importance of early testing, defect clustering and the pesticide paradox, and state that testing should always be context dependent.

The first principle stresses the importance of early testing. This principle states that testing should begin as early as possible in the software development life cycle (SDLC).   
  
If defects are detected during the initial stages of the SDLC, they can be corrected with minimum effort in terms of cost and time. Correcting defects during the initial stages prevents defects from cascading onto the Development and Implementation stages.   
  
Just before the release of the software, the pressures of a deadline could lead testers to compromise their testing efforts. Software testing should not just be a part of development but should be carried out throughout the SDLC. This eliminates the problems associated with trying to accommodate everything into a single testing effort.

If an error is detected during the Analysis phase through static tests, then there is little chance of an error being introduced during the Design phase.  
  
  
On the other hand, if there is no testing conducted during the Analysis phase, errors can only be discovered during the Implementation phase. During implementation you may realize that there is a serious design issue with the product which will be costly to fix.

During the Design phase, if a defect in the code goes undetected, it can result in further defects. Defects can have a tendency to replicate themselves in various parts of the code. A considerable amount of rework is required to correct such defects.

If a defect is uncovered during Acceptance testing at the Implementation stage, it means going right back to the Analysis phase to review specification documents. This is important to judge whether you have understood the customer requirements. It can also involve reworking specifications and retesting them resulting in increased costs.   
  
Extensive studies are carried out to study the effect of defects on cost at various stages of the SDLC. Testing objectives differ at every stage of the SDLC. The objective for testing at the Analysis stage is to review the software requirements document, and at the Development phase, the objective is to test the code by executing it.  
  
It is impossible to put accurate figures to represent the difference in costs required to correct defects at different stages of the SDLC. The accompanying graph attempts to present a rough idea of the same. The graph represents the cost escalation model. It shows that the later an defect is detected, the more it costs to fix.

Graphic

*The graph represents a steady increase in the cost of correcting defects at various stages of the SDLC. The X axis shows the various testing stages of the SDLC, namely Requirements, Coding, Program Testing, Acceptance Testing, and Live Use. The Y axis shows the increase in costs from zero dollars to 500,000 dollars. The increase in cost is shown by a curved line which begins at zero dollars and ends at 500,000 dollars on the Y axis and at the Live Use stage on the X axis.*

The second principle of applied testing is based on the pretext that the defects aren't spread uniformly in the software. During pre-release of the software, a few software modules will display the most number of defects and operational failures. This is referred to as defect clustering. This can be due to

* inexperience of the development team
* past experiences of the development team
* cascading effects of new changes on various parts of the code
* volatile code, which is more susceptible to defects
* complex logic and structure of a software application

Defect clustering is based on the Pareto principle. The Pareto principle states that about 80% of defects will be found in approximately 20% of the modules.  
  
You should test components of the software that are more susceptible to defects. However, this doesn't mean that you should disregard less susceptible areas. You should balance your testing efforts so that both types of software components are given due attention.

The third applied principle of software testing discusses the Pesticide Paradox. This paradox states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs. This can happen because during the debugging and testing, the software might have undergone changes. These new changes can no longer be checked using an old test case, so it becomes important to revise existing tests, and develop newer ones which will be able to uncover more bugs.  
  
For example, if a change is introduced in the code, new tests need to be developed to check whether the change has any unwanted effects on other components of the software. This is referred to as *regression testing.*The same set of regression tests might not be able to uncover defects during production because the testing objective might have changed altogether.

The fourth applied principle of software testing states that you need to vary testing efforts depending on the circumstances. For example, you create a plan for a web site which stores confidential medical information. This information can only be viewed by doctors, so the test plan for this web site will be vastly different from a test plan created for a large online bookstore that sells to the general public.  
  
  
A different plan would be required for various reasons. The bookstore will need to tie in with complex logistical systems based around shipping product in a timely fashion. It may also have many more users than the medical site. The key job of medical site, however, could be viewed as controlling access to the information it houses.

Question

During development of an e-commerce web site, the specifications of its checkout system were reviewed. The review revealed a flaw in the logic of the system. As a result, the specification was rectified before the web site specifications were sent for development. Which principle of software testing has the development team based their testing efforts on?

**Options:**

1. Defect clustering – Principle Two
2. Pesticide paradox – Principle Three
3. Early testing – Principle One
4. Testing is context dependent – Principle Four

Answer

***Option 1:****Incorrect. The second principle of applied testing is based on the pretext that the spread of defects is not uniform. And, during pre-release, a few software modules will display the most number of defects and operational failures compared to others. In this scenario, the development team has followed the principle of early testing that states that it is always beneficial, in terms of time and cost, to begin testing in the initial phases of the SDLC.*

***Option 2:****Incorrect. The third principle of applied testing discusses the Pesticide Paradox. The principle states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs. The development team has based their testing efforts on the principle that effective testing always begins during the initial phase of the SDLC. This corrects defects during Analysis and Design and prevents them from appearing during development and implementation.*

***Option 3:****Correct. The first principle of applied testing stresses the importance of early testing. The principle states that testing should begin as early as possible in the SDLC. Early testing saves efforts in context to both time and cost.*

***Option 4:****Incorrect. The fourth principle of applied testing states that testing efforts will vary in different circumstances. For example, testing for a e-commerce site will be very different from testing for safety-critical software. In this scenario, the development team has followed the principle of early testing that states that it is always beneficial, in terms of time and cost, to begin testing in the initial phases of the SDLC.*

**Correct answer(s):**

3. Early testing – Principle One

Question

Your organization is developing a content management system. This system uses separate modules to run most of its core functions. During component testing, you discovered that the user authentication module has a high number of defects. The same module also faced problems during integration testing. Which principle of software testing is proved by the above scenario?

**Options:**

1. Defect clustering – Principle Two
2. Early testing – Principle One
3. Pesticide paradox – Principle Three
4. Testing is context dependent – Principle Four

Answer

***Option 1:****Correct. The first applied principle stresses the fact that defects are not spread uniformly in the software. Some components of the software display more defects and operational failures compared to others.*

***Option 2:****Incorrect. The first applied principle is based on the fact that you should start testing from the initial phases of the SDLC. As the initial phases primarily involve static tests, early testing saves efforts in context to both time and cost. This scenario illustrates the concept of defect clustering which indicates that certain software modules are more susceptible to defects than others.*

***Option 3:****Incorrect. The third applied principle discusses the Pesticide Paradox which states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs. This scenario illustrates the concept of defect clustering which indicates that certain software modules are more susceptible to defects than others. More susceptible modules need to be resolved earlier than less susceptible modules.*

***Option 4:****Incorrect. The fourth applied principle states that you need to change your testing efforts based on different circumstances. For example, testing for a gaming site will be very different from testing for an online store selling music CD's. This scenario illustrates the concept of defect clustering which indicates that certain software modules are more susceptible to defects than others.*

**Correct answer(s):**

1. Defect clustering – Principle Two

Question

You have a comprehensive set of tests developed for a spreadsheet program developed by your company and you apply these tests to each successive version of the program that is released. In the most recent version very few defects were revealed by your tests and you take this to mean that the software is now very refined. Which principle of software testing might prove your perception wrong?

**Options:**

1. Early testing – Principle One
2. Defect clustering – Principle Two
3. Pesticide paradox – Principle Three
4. Testing is context dependent – Principle Four

Answer

***Option 1:****Incorrect. The first principle of applied testing discusses the importance of early testing. This principle states that testing should begin as early as possible in the SDLC. During the initial phases of the SDLC, static tests are used to correct defects. Static tests are more economical than dynamic tests required to resolve errors in the later stages of the SDLC. The pesticide paradox will prove that your perception of using standardized tests to resolve defects is an incorrect approach to correct new bugs.*

***Option 2:****Incorrect. The second principle of applied testing is based on the pretext that the spread of defects is not uniform. And, during pre-release, a few software modules will display the most number of defects and operational failures compared to others. You should have paid attention to the pesticide paradox that states that you should always revise your testing efforts to resolve new defects.*

***Option 3:****Correct. The third principle of applied testing discusses the Pesticide Paradox which states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs.*

***Option 4:****Incorrect. The fourth principle of applied testing states that testing efforts will vary in different circumstances. For example testing for an e-commerce site will be very different from testing for safety-critical software. The pesticide paradox will prove that your perception, that using standardized tests to resolve defects, is an incorrect approach to correct new bugs.*

**Correct answer(s):**

3. Pesticide paradox – Principle Three

Question

Your company produces noncritical office software. A new tester has recently joined your team. This tester had previous experience in testing military systems where failures would be particularly hazardous. In his first week, the new tester discovers a fair number of defects, but he seems much slower than your other testers because he insists on fully exercising any code that he is given. Which principle of software testing might this new employee benefit from learning?

**Options:**

1. Early testing – Principle One
2. Defect clustering – Principle Two
3. Testing is context dependent – Principle Four
4. Pesticide paradox – Principle Three

Answer

***Option 1:****Incorrect. The first principle of applied testing stresses on the importance of early testing. This principle states that testing should begin as early as possible in the SDLC. Early testing saves efforts in context to both time and cost. Testing efforts are always relevant to the context they are being used in. As this tester is used to testing complicated logic and structure, he would tend to waste a lot more time for testing simple software.*

***Option 2:****Incorrect. The second principle of applied testing is based on the pretext that the spread of defects is not uniform. And, during pre-release, a few software modules will display the most number of defects and operational failures compared to others. This scenario highlights the principle that states that testing is always context dependent. This tester is suitable for testing complicated software and would not be suitable to test software that uses simple logic.*

***Option 3:****Correct. The fourth principle of applied testing states that testing efforts will vary in different circumstances. For example, testing for an e-commerce site will be very different from testing for safety-critical software, so using this tester in a simple testing environment would be a waste of time and cost.*

***Option 4:****Incorrect. The third principle of applied testing discusses the Pesticide Paradox which states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs. Testing efforts are always relevant to the context they are being used in. As this tester is used to testing complicated logic and structure, he would tend to waste a lot more time for testing simple software.*

**Correct answer(s):**

3. Testing is context dependent – Principle Four

**2. Summary**

There are four applied principles that guide testers while devising software testing plans.  
  
The first principle stresses that testing activities should start as early as possible in the software or system development life cycle. Principle two shows that a small number of modules contain most of the defects discovered during pre-release testing. The third and fourth principles outline the Pesticide Paradox, and state that testing is context dependent.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

The Importance of Software Testing and Mitigating Harm

Learning Objective

*After completing this topic, you should be able to*

* *recognize the fundamentals of software testing*

**1. Exercise Overview**

You need to train the new testing team about how software testing improves quality, how early testing proves economical, and how testing principles apply to scenarios presented.

You need to train the team on:

* how testing can help improve the quality of software
* the importance of early testing
* how testing efforts differ depending on context

**2. Recognizing how testing improves quality**

You're a test manager in an organization. A development team within your organization has created software that manages railway line switches to control railway tracks and the train schedule. The scenario involves very real safety concerns. If the software fails, it could result in hundreds of deaths and millions of dollars of damage.

Question

The success of the software is important as the slightest defect might end up putting the lives of millions at risk.  
  
What should the testing team focus their testing efforts on?

**Options:**

1. Functional attributes of the software
2. Functional and non-functional attributes of the software
3. Exhaustive testing
4. Risk assessment and management

Answer

***Option 1:****Incorrect. Although testing for functional attributes ensures that the product performs an expected task seamlessly, it is also important to test non-functional attributes to ensure that the task is completed in the least amount of time and effort. You would need to test for non-functional attributes too.*

***Option 2:****Correct. Testing for functional attributes ensures that the product performs an expected task. Testing for non-functional attributes measures how well or fast a task is performed. Testing non-functional attributes usually requires a metric, such as time to complete a task.*

***Option 3:****Incorrect. Exhaustive testing checks all combinations of input values and preconditions. Exhaustive testing, although ideal, is neither practical nor advisable. It would be a good idea to concentrate on risk assessment, and the parameters of validation and verification.*

***Option 4:****Correct. A test approach should be devised by juxtaposing the test with the risks which the customer, the project stakeholders, the project as an entity, and the software is exposed to.*

**Correct answer(s):**

2. Functional and non-functional attributes of the software  
4. Risk assessment and management

**3. Understanding early testing**

Early testing and defect removal is usually much less expensive than repairs later in development. This is especially true in a complex physical system that may need to be taken offline for testing once it is constructed.

Question

You plan to brief your team on the testing plan they need to create.  
  
At which stage of the SDLC would you recommend testing to begin?

**Options:**

1. Design
2. Development
3. Analysis
4. Implementation

Answer

***Option 1:****Incorrect. Testing that begins at the Design stage only tests the design of the software and does not check software documentation. Although defects detected at the Design stage can be easily solved, ideally, testing should always begin at the Analysis stage.*

***Option 2:****Incorrect. Testing that begins at the Development stage, might require changes in code. Although not very difficult to implement, defects of the Design and Analysis stages might pass unnoticed. Thus, ideally, testing should always begin at the Analysis stage.*

***Option 3:****Correct. Testing should ideally begin at the Analysis stage. If all clarifications are answered at this stage, there would be a very slim chance of a defect cascading onto the later stages of the SDLC, thus saving time and money.*

***Option 4:****Incorrect. Defects detected during Implementation are very difficult to correct and involve a huge amount of investment both in terms of money and time. It could also mean going right back to the Analysis stage and recreating the product. You should always begin testing during the Analysis stage.*

**Correct answer(s):**

3. Analysis

Question

Static and dynamic tests are used at various stages of the SDLC. You need to inform the team on what methods they need to utilize at what stage.  
  
Match each testing type with its relevant action.

**Options:**

1. Static
2. Dynamic

**Targets:**

1. Analyzing client requirement documents
2. Reviewing software design documents
3. Reviewing source code of the software
4. Executing the code of the software

Answer

*Static methods comprise reviewing software related documentation. Static tests occur at the Analysis and Design stages of the SDLC. Dynamic testing includes running the code of the software.*

*Static methods comprise analyzing software related documentation, like specification documents and client requirements. Dynamic testing includes running the code of the software.*

*Static methods comprise reviewing software related documentation and source code. The code is not executed as is done with dynamic testing. Reviewing code generally happens during the Development stage of testing.*

*Dynamic testing happens during Development and Implementation when the testing team executes the code of the software to check whether the product is performing as expected. Static testing is generally comprised of reviewing software related documentation and source code.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option A

Target 3 = Option A

Target 4 = Option B

**4. Explaining testing principles**

You want to inform the team that software testing is based on certain principles.

Question

The spread of defects in software is not uniform and a few software modules will display the most number of defects.  
  
Which principle of software testing is this?

**Options:**

1. Defect clustering
2. Pesticide paradox
3. Exhaustive testing is impossible

Answer

***Option 1:****Correct. Defect clustering is based on the Pareto principle. The Pareto principle states that about 80% of defects will be found in approximately 20% of the modules.*

***Option 2:****Incorrect. The Pesticide Paradox states that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs.*

***Option 3:****Incorrect. This principle states that exhaustive testing is rarely achievable in practice.*

**Correct answer(s):**

1. Defect clustering

Question

You have created a test plan for the first release of audio recording software. You use the same plan to test the software for its second and last releases. The tests reveal very few defects and you decide that the software can be released without further rounds of testing. After the release, some bugs come to light.   
  
Awareness of which principle may have reduced the number and severity of these bugs?

**Options:**

1. Early testing
2. Defect clustering
3. Pesticide paradox
4. Testing is context dependent

Answer

***Option 1:****Incorrect. This principle states that testing should begin as early as possible in the SDLC. Early testing saves efforts in context to both time and cost. The principle which outlines the concept of the pesticide paradox is more applicable in this scenario.*

***Option 2:****Incorrect. Defect clustering states that the spread of defects in software is not uniform and a few software modules will display the most number of defects. In this case, the relevant principle is the pesticide paradox.*

***Option 3:****Correct. The Pesticide Paradox states that if you do not develop new tests and attempt to use a standard set of tests for all release of software, the tests will cease to uncover bugs.*

***Option 4:****Incorrect. Testing effort varies in different circumstances. For example, testing for an online shopping store site will be very different from testing for safety-critical software. You are violating the principle that discusses the concept of the pesticide paradox.*

**Correct answer(s):**

3. Pesticide paradox

Question

During the development of an online bookstore, the specifications of the search functionality of the software were reviewed. During these reviews, a defect in the design logic and functionality of the software was discovered. This defect was corrected in the specification and the software was sent for development. If these defects had made it though to a later stage of the software lifecycle, they could have been more time consuming and costly to remove.  
  
Which principle of software testing did the development team follow?

**Options:**

1. Early testing
2. Defect clustering
3. Pesticide paradox
4. Testing is context dependent

Answer

***Option 1:****Correct. This principle states that testing should begin as early as possible in the SDLC. Early testing saves efforts in context to both time and cost as it involves only static testing techniques.*

***Option 2:****Incorrect. The spread of defects in software is not uniform. A few software components will always display more defects and operational failures compared to others. The principle of early testing in the SDLC is more applicable in this scenario.*

***Option 3:****Incorrect. The Pesticide Paradox stresses that if you continue using the same set of tests over and over again, the tests will cease to uncover bugs. The test team has followed the principle of early testing in this scenario.*

***Option 4:****Incorrect. Testing efforts will vary in different circumstances. For example, testing for a content management system will be very different from testing for an online store that sells medical equipment. The test team has followed the principle of early testing in this scenario.*

**Correct answer(s):**

1. Early testing

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

The Fundamental Software Test Process

Learning Objectives

*After completing this topic, you should be able to*

* *recognize the test process in a given project*
* *identify the component steps in the test process*

**1. The software testing process**

Once a software application is developed, it is important to test it and verify whether the software has met the purpose and the objective for which it was developed. The more rigorous the test process, the better the product quality.   
  
Although testing rigor may vary across software products – products with fewer risks may not require as much testing as a product with higher risks – the software testing process remains the same.

This process includes distinct phases where one phase follows the other and where phases are interdependent on one another.

Note

*Although the process is depicted here as strictly linear, there will be times when you need to go back to an earlier task, or run tasks in parallel.*

There are five fundamental phases in the software testing process.

Graphic

*Description of the software testing process flow chart:  
The five phases of the software testing process begins with test planning and control as the first phase. It then moves to the second phase – test analysis and design – and then the test implementation and execution phase. This is followed by the evaluating exit criteria and reporting phase, with the process finally ending with the test closure activities phase.  
Description ends.*

**Test planning and control**

In the test planning phase, you establish the need or mission of testing. This helps you focus on what needs to be tested, the scope of testing, and what defines the exit or completion of the test. These details are outlined in the test plan – a deliverable of this phase.  
  
The test plan also specifies the schedule of activities, resources required for the testing process, and the test environment. Additionally, it also lists any risks that might affect test execution and documents contingency plans.  
  
Test control compares actual progress against the test plan. Reports are the deliverable of test control which include deviations from the plan. Test control involves taking action to meet project objectives, and is an ongoing activity during the project.

**Test analysis and design**

In the test analysis and design phase, test objectives determined during planning are converted into plausible test conditions and test cases. During this phase, you review the test basis (including requirements, architecture, design and interfaces), and evaluate the testability of the test basis and test objects. You also identify and prioritize test conditions based on the analysis of test items, the specification, behaviour and structure.  
  
In this phase, you also design test cases and prioritize them, identify the test data necessary to support the test conditions and test cases, and design the test environment set-up and identify the tools and infrastructure which will be required.

**Test Implementation and execution**

The test implementation and execution phase is where test procedures, or scripts, are specified, the test environment set up, and tests are run. Test implementation includes developing and prioritizing test cases and procedures, creating test data and creating test suites from the test procedures to ensure efficient test execution. You should also verify that the test environment is set up properly.  
  
Test execution is the phase where you execute tests – manually or by using execution tools – in the sequence planned. You also log the outcome of test execution, record what software and version is under test, and what test tools and testware are being used. You also compare actual results with expected results and report any discrepancies as incidents. These incidents will be analysed in order to establish their cause – was it a defect in the software, a defect in the test data, or maybe a mistake in the test execution? Finally, test activities are repeated for any discrepancy. In the case of confirmation testing, you would re-execute a test that previously failed to confirm a fix. In the case of regression testing, you might execute a corrected test in order to ensure that a defect has not been introduced into an unchanged area of software.

**Evaluating exit criteria and reporting**

In the exit criteria evaluation and reporting phase, you analyze the test execution results against the objectives defined in the analysis and design phase.  
  
You determine if the exit criteria – conditions determined for test completion – are met. You then analyze if more tests are required, or if the exit criteria set should be changed. You also prepare a test summary report for the stakeholders.

**Test closure activities**

Test closure activities – the final stage of the testing process – require you to consolidate and document data from the completed phases of testing.  
  
This phase involves checking deliverables and ensuring that all test incidents are closed. During this phase, testware – scripts, test environment, and other test infrastructure – are archived for future reference before the product is handed over to maintenance testers for regression testing.

The five-step software testing process ensures that the developed software meets the stated objectives and mitigates any risks of failures that may occur due to the defects in the software. Although this process follows a sequence of activities, it is not rigid. Some steps in the process are iterative and can be conducted in parallel for rigorous and extensive testing.  
  
For example, suppose you completed the planning phase for a software application and have begun designing test activities for it. A change in the market requirements for the software is reported, which in turn modifies stakeholder requirements and project objectives. In this case, you need to revisit the planning stage after the design phase. Similarly, you also iterate the analysis phase, after test implementation and exit criteria evaluation, if there are changes in the test plan.

Graphic

*In this depiction of the software testing process, the test planning and control phase iterates after the test analysis and design phase, the test analysis and design phase iterates after the test implementation and execution and evaluating exit criteria and reporting phases.*

Question

Sequence the phases constituting the fundamental software testing process from start to finish.

**Options:**

1. Analyze test goals and prepare a test plan
2. Analyze test conditions and architecture and then create test cases
3. Implement test suites and report test incidents
4. Evaluate the exit criteria and prepare test summary
5. Archive testware and evaluate test results

Answer

**Correct answer(s):**

**Analyze test goals and prepare a test plan is ranked**

In the software testing process, the first phase is to analyze the test goals, objectives, and the risks, and plan how testing will proceed. This phase is called test planning and control.

**Analyze test conditions and architecture and then create test cases is ranked**

The second phase of the software testing process is where testing objectives are converted into test conditions and cases.

**Implement test suites and report test incidents is ranked**

Implementing test suites and executing them is the third phase. Implementation activities include creating test scripts. Execution is where you run tests and record incidents.

**Evaluate the exit criteria and prepare test summary is ranked**

The fourth stage of the software testing process is to evaluate the exit or test completion criteria defined during planning. You validate these criteria against the actual test results documenting them in a test summary report.

**Archive testware and evaluate test results is ranked**

Test closure is the final phase and involves analyzing test results for future projects and archiving testware for maintenance testers.

Suppose you're a test manager in an organization developing an online auction web site for automobiles. You want to test the developed software to evaluate its effectiveness, and to check if it meets the stated expectations.

Applying the fundamental software test process, you begin with the planning phase. In this phase, you ensure that you understand the goals and objectives of the stakeholders – your company, the clients, and the end-users.   
  
The objective of the auction web site is to provide an online portal that allows users to log in and place bids on automobiles, with the web site allowing the highest bidder to proceed with the transaction. Because the transaction is financial, user privacy and security is paramount.  
  
These objectives help you identify what elements are critical to the web site and let you decide areas that need to be tested. In this example, the testing focus will be ensuring that only authorized users with valid logins can perform a transaction, checking if users can place bids, ensuring that the web site allows only the highest bidder to access the transaction page, and determining that a secure connection is provided for complete anonymity during transactions. You can then continually monitor these objectives throughout the testing process.

During the planning phase, you also determine the kinds of tests to run and the people who will be involved. You also create a schedule and then define the exit criteria based on the areas to be tested.   
  
For your auction web page, you will set 80% statement coverage throughout the web auction system and 100% branch coverage for areas of code relating to financial transactions as the exit criteria against which test results are validated. Testing is deemed complete only if the exit criteria are met.

With a test plan with exit criteria and schedules in place, you proceed to the analysis and design phase, where you

**review the test basis and identify test conditions**

Your first task in the analysis phase is to review the test basis and identify test conditions. While reviewing the test basis, you analyze risks, requirements, and specifications. Based on these, you identify test conditions for your software.  
  
In the auction web page example, you want users to be able to view automobile models and place bids without being impacted by large volumes of site traffic. These become your test conditions, where you can check if the web page functions as expected during hours of peak network traffic.

**design tests and evaluate testability**

Based on the test conditions, you design the tests and evaluate the testability of the component – the auction web page, in this example.  
  
If a requirement for the auction web page is that the web page should respond quickly, you must define 'quickly' and set parameters for how quick you want the page to respond – five or ten seconds, for example. Accordingly, your test environment will include resources that allow this test to occur.

Once tests are designed and a strategy put in place, you develop and execute test cases.  
  
To do this, you create test procedures – instructions for test cases. These procedures focus on checking elements you defined as critical earlier. These elements could include, for example, whether you can log into the site, whether an automobile name entered in the search string displays appropriate images, and whether you're able to place a bid on the car. These details are mentioned in the test cases.

After creating test procedures and cases, you create test suites, which are logical collections of test procedures. For your auction web page, the collection of test cases that check whether users are able to log in becomes one test suite.  
  
In addition, you also ensure that the test environment is set up, and use it to run possible trial tests to measure the efficiency of the environment.

In the test execution phase, you execute the test suites and document test outcome logs. While documenting, you record the versions of the software under test, your test tools, and testware. You also compare actual results to expected results and, where there are discrepancies, record incidents.  
  
Suppose currency is not displaying correctly in your web page during testing. As a result, transactions become tedious for a user, who is unable to view the amount accurately. This instance is reported as an incident during the execution phase.

In this phase, incidents or discrepancies reported should be fixed, and test activities should be repeated for each discrepancy to confirm that the fixes function correctly and haven't altered the program in other ways. This procedure may be repeated until test results display expected outcomes.

After running the test suites, you need to determine whether the exit criteria set during the planning phase are met. This is the exit criteria evaluation and reporting phase where you check logs of the test outcome against the exit criteria to determine if they are met. Wherever a criterion is not met, depending on how critical it is, you may revise it before re-running the tests.   
  
For example, in the auction web page, you can revise the criteria for the 80% statement coverage to 75% and run tests again. But you would not revise the 100% financial coverage, because it is directly connected to ensuring a secure finance system. In this case, you devise further tests to check 100% financial coverage.

Note

*Testing is considered complete when exit criteria are met.*

Once exit criteria are met, you can create a test summary report. This report informs the stakeholders about the extent of testing undertaken and its outcomes.

In the final stage of the testing process – test closure – you deliver the tested software and document whether incident reports are resolved or deferred. Categorizing incidents is important, because deferred incidents might become requests for a future version of the software. For example, in your auction web page, deep bookmarking might not be available for individual cars in the version, because it is not critical for successful implementation of the application. This feature, however, is user friendly and could be made available in the next version.

In this phase, you also archive the scripts and details of the test environment so that it is readily available for use at a later stage. As a sign-off, you hand over the tested software to the maintenance team in charge of deploying and maintaining the web auction page. Lastly, you document any lessons learned during testing, in an overall project evaluation report, so that these lessons can be applied to future projects.

Question

As a test engineer, you want to test the software for managing payroll services in your organization. The focus of this payroll portal is to maintain employee privacy and security of transactions. Rank the tasks in the order you would execute them in the software testing process.

**Options:**

1. Identify unauthorized user access as a security risk
2. Specify as a test condition that only users with authorized user names and passwords can log on
3. Create a test procedure for the ability to log into the payroll portal
4. Check the exit criteria defined for security of transactions against the test outcome
5. Archive testware and prepare a project evaluation report

Answer

**Correct answer(s):**

**Identify unauthorized user access as a security risk is ranked**

Understanding the goals of your project and identifying the risks belong to the planning phase of the testing process. This also helps you determine the tests required to meet the test objective.

**Specify as a test condition that only users with authorized user names and passwords can log on is ranked**

Setting up test conditions is done in the analysis and design phase. You also proceed to design the test cases, and identify test data required to support those cases.

**Create a test procedure for the ability to log into the payroll portal is ranked**

You create test procedures and execute test suites during the implementation and execution phase of the process. In this phase, you also repeat the test activities designed for fixed defects, to validate these fixes.

**Check the exit criteria defined for security of transactions against the test outcome is ranked**

Evaluating the exit criteria is the fourth phase of the testing process. You compare the actual results against the estimated results during this phase. You do this using the logs of the test execution.

**Archive testware and prepare a project evaluation report is ranked**

During the test closure phase, you archive the testware for future test components and hand it over to the test maintenance team. In addition, you prepare a test summary report.

**2. Summary**

Software testing is an extensive process that includes a number of phases and tasks. This process includes five fundamental phases – test planning and control, analysis and test design, test execution and implementation, exit criteria evaluation, and test closure activities.  
  
Each of these phases includes specific tasks. Planning requires you to define the test objectives. Control compares actual progress with the test plan on an ongoing basis, and takes action where necessary. Analysis and design is where testing objectives are converted into test conditions and cases. Implementation includes designing test scripts, and execution is where the test scripts are executed. Evaluating exit criteria and reporting is where test execution is assessed against the objectives. Test closure activities include creating a test summary report and archiving testware.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

Exit Criteria and Test Closure Activities

Learning Objective

*After completing this topic, you should be able to*

* *identify the two final steps of the test process in a given scenario*

**1. Exit criteria and test closure**

The objective of testing is to ensure that a quality product that meets the purpose for which it is created is released. Based on this objective, test plans are created and exit criteria are defined, which further impacts how tests are designed and executed. Only when these exit criteria are met is testing considered to be complete.   
  
Evaluating exit criteria is an important task or phase in the test process. Exit criteria are a set of business-driven conditions that determine the customer-ship or deployment decision for a software application.

Exit criteria are defined during the planning phase of the test process. These criteria act as a set of checkpoints that provide focus on what you need to track during testing and also what you should validate your test outcome against.

For example, you plan to test a web application that helps you book airline tickets. You define exit criteria based on the critical factors of the application that need to be fully tested. A critical element of this application is to test its functionality – for example, selecting a flight and being able to enter the number of tickets to book in that flight.   
  
So you set an exit criterion that checks 70% of the application functionality with a certain rigor – no critical defects reported for a particular function, for example. This criterion helps determine what needs to be evaluated after testing. The testing is considered complete only if this criterion is met.

Meaningful exit criteria that are measurable and achievable help

* ensure test efficiency and effectiveness
* prioritize and execute testing activities
* evaluate current activities in the testing process and decide steps for further action
* suggest corrective measures for incomplete and faulty efforts
* identify plausible risks for your test project

Evaluating exit criteria lets you check whether the conditions to complete the testing process are met. Evaluation requires you to analyze the test results of a completed test suite against the objectives and the criteria set during the planning phase.

In exit criteria evaluation, you analyze different types of criteria – all of which are part of the exit criteria – such as the

**coverage criteria**

Coverage criteria help you decide test cases that must be included during the exit criteria evaluation process.  
  
For example, when testing an application that helps you book air tickets; your coverage criteria could be to test the functionality of the application, which enables the user to select the required airline tickets and the mode of payment. You will not cover features that process credit card numbers in this test suite.

**acceptance criteria**

The acceptance criteria enable you to check whether the software under testing has passed or failed in the overall process.  
  
For this test suite example, the acceptance criteria could be to pass the application only when it allows you to select the required flight service and the date of travel. The exit criteria for this process could specify that testing is complete only when you have completed every step of the ticket-booking process and prepared the test report.

If you're checking a spreadsheet application, you might set a number of exit criteria statements to check if testing is complete. The following serve as an example and should not be taken as complete, accurate or exhaustive:

* final draft of the product documentation, including the Contents and Help files, are reviewed, tested in QA, and approved by the core team
* help files are launched with the print option when the required menu option is selected
* menu options available on the UI of the spreadsheet are enabled and highlighted when rolled over
* spreadsheet runs on all supported hardware and software configurations, as mentioned in the documentation files
* defects of all priorities are fixed and closed during regression testing
* navigation across different instances of the spreadsheet is enabled

In addition to analyzing the exit criteria, the exit criteria evaluation phase includes a number of tasks and activities of which three are particularly important.

**Check test logs**

When you begin exit criteria evaluation for your test suite, you collect and check the test logs acquired from the test implementation and execution phase. You check these logs against the exit criteria set for your test suite during the planning phase. You then identify evidence for every test executed and the defects that are logged. Defects are analyzed to distinguish between those that are resolved and those yet to be confirmed or fixed.

**Estimate additional test requirements**

Depending on the test log analysis, you can estimate if additional tests are required in a test suite for a more thorough checking. You may decide this based on the number and the criticality of defects logged. At times, you may even revise exit criteria.   
  
Additional tests are required when the designed tests do not meet the required test coverage. You change the exit criteria only when the business or technical risks of the software under testing changes. However, these decisions require the approval of stakeholders.

**Prepare test summary report**

After analyzing the test logs and assessing if more tests are required, you prepare a summary report for the stakeholders of the project. This document communicates the test outcome to your stakeholders, which enables them to make critical decisions about the software under testing. This document summarizes testing activities and an evaluation of the test results against the exit criteria.

After you evaluate the exit criteria for your testing process, you proceed to the final phase of the testing life-cycle – test closure. This phase ensures the completion of all test activities and the sign-off of the end product. Once you reach this phase, you're ready to deliver the software to your client.

Note

*In tests executed for research purposes, you close testing as soon as you've gathered the required information, because complete testing could exceed the scope of your requirements.*

Test closure activities include

**checking deliverables**

Test closure is initiated by checking planned deliverables against the actual deliverables meant for the client. This helps check if incident reports are either resolved or deferred according to the planned deliverables. Planned deliverables specify if an identified incident is to be fixed in the current version of the software. For example, in the spreadsheet application, an exit criterion is compatibility with third-party products. If the spreadsheet is incompatible with a specific third-party product, then you check your planned deliverable to see if this compatibility is required. If not, you can defer fixing this issue, documenting it so that it's addressed in the future versions of the application.

**archiving testware**

After checking deliverables, you collate and archive testware – test scripts, test environment components, and any tools used in the test infrastructure. You archive testware so that the same components can be reused during maintenance testing, making the process time- and cost-effective. Archiving also helps you compare test results between the different versions of the software.  
Suppose an earlier version of the spreadsheet application had been tested by you and the test logs of that test are available from the test library. You can compare these with the current test logs to check if any errors are repeated. You can then archive your test logs for future use.

**submitting testware to maintenance team**

Submitting testware to the maintenance team is an important activity of test closure. The maintenance team – independent of the test team – supports the software and is responsible for defect resolution after the application goes live. Archived testware in the test library helps maintenance testers reuse the same test scripts and environment.

**evaluating the overall test process**

Finally, you evaluate the overall process that was executed to test the software. This helps identify and assess lessons learned from the test process, letting you improve future test processes.

Question

As a development tester, you've recently completed multiple test suites for a word processing application to check if 85% of statement coverage is tested. You also checked to ensure that user-defined words could be added to the dictionary.  
  
Having implemented the test suites and generated the test logs, what activities would you perform as the final steps of the test process?

**Options:**

1. Check if the functionality of the word processor is tested up to the required degree of test coverage as planned
2. Create exit criteria to check that 85% statements are covered and words to the existing dictionary can be added without errors
3. Validate test logs against the exit criteria and identify if further tests are required

Answer

***Option 1:****Correct. You check the actual deliverable against the planned deliverable during test closure phase before delivering the tested software to the client. At this stage, you ensure that all incident reports and defects are resolved.*

***Option 2:****Incorrect. You create the exit criteria that you require for testing, during the initial planning phase. You would check these criteria against the test outcome, analyze whether they are met, and if met, exit testing.*

***Option 3:****Correct. Once test logs are generated, it is important to validate them to evaluate exit criteria. At this stage, if you determine that exit criteria are not met, you can identify further tests to cover the exit criteria.*

**Correct answer(s):**

1. Check if the functionality of the word processor is tested up to the required degree of test coverage as planned  
3. Validate test logs against the exit criteria and identify if further tests are required

**2. Summary**

In the software testing process, the last two phases – evaluating exit criteria and reporting and test closure – help you assess whether testing is complete and document the test results and the scripts for future use.  
  
Evaluating exit criteria and reporting requires you to check the test logs acquired from test execution and then compare them against the exit criteria set during test planning. In addition, you also analyze if any further tests are required and prepare the test summary report.  
  
During test closure, you check the actual deliverables against the planned deliverables, before you deliver the software to the client. You then archive testware in the test library, which is used by the maintenance testers. Finally, you analyze and suggest improvements to the overall test process documenting them in the project evaluation report.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

The Psychology of Software Testing

Learning Objectives

*After completing this topic, you should be able to*

* *identify levels of independence in testing*
* *recognize the role of communication in the psychology of software testing*

**1. Levels of independence in testing**

Among the many factors that affect software testing, psychological factors are important, because they influence the success of the testing activity. How an individual identifies and reports defects, how evidence is gathered and shared, and how results are communicated and interpreted, are all factors determined by the psychological make up of individuals involved.   
  
  
The psychological make up of a developer differs from that of a tester, and consequently influences the process of testing.

Testing requires an objective mindset. This objectivity exists only if the person testing the product is not personally attached to the product.  
  
For example, when a developer creates an application, he constructively resolves the issues in the design to ensure that the end product meets the required standards. But while testing the product, the developer may miss defects because he may approach the product – given the effort he put in and the personal involvement in developing the product – as one without defects.

But a tester, who is not involved in software development, stays detached from the product being tested. This helps him or her objectively assess the product and find defects because he or she consciously looks for them during testing.

Testing activities should be assigned to someone who was not part of the software development process. This ensures separation of responsibilities between a tester and a developer, called *independence*.

Independence in software testing introduces objectivity and adds value. It defines the success of the software testing process by preventing author-induced biases from creeping in and allowing valid defects to be reported.

You can implement different levels of testing independence at different stages of the development life cycle depending on the nature of the testing and reviewing activities in that stage.

There are four basic levels of independence – graded from high to low – in software testing. The grade of independence depends on the degree of responsibilities shared between the members of the testing and the development teams.

**Software developer**

When the developer who created the software tests his own software, the level of testing independence obtained is minimal. There is no actual separation of duties, because the tester and the developer are the same person. This first level of independence is the lowest in the order of independence and can be employed for cursory reviews or when the project is considered low risk.  
  
For example, when you program a word-processing application, you can do a self-review of the product for any design-based defects easily identifiable by you. But this level of independence would not suffice for in-depth analysis of the software – for example, to check functionality defects or third-party application support.

**Peer reviewer**

The second level of testing independence is when a peer within the team reviews the product or application. This person brings a fresh perspective to the testing and tends to be more objective than the product author. Typically, peers are members from the same development team, who may be in charge of developing a different application or different portions of the same application.

**Internal tester**

For a higher level of testing independence with improved objectivity, you can employ an internal tester from a different functional group or a testing team. For example, you can ask a qualified tester from the QA team in your organization to review your software.  
  
This is the preferred and most common level of testing independence employed by most organizations. It is cost effective, because you employ someone within your organization to independently evaluate your software.

**Third-party reviewer**

The highest and the fourth level of testing independence is achieved when a third-party reviewer – usually a certified tester from a different organization – tests your software. Although expensive, this level provides optimum results in a test scenario, providing maximum separation of responsibilities between the development and testing teams.  
  
Suppose your team develops a calendar application. To ensure complete objectivity, you can hand over the product to a third-party test team. This team approaches the product as flawed, unlike teams within the organization which may not bring this level of objectivity. As a result, they're able to find defects that teams within the organization may overlook.

Although objectivity is critical during software testing, sometimes developers test their products due to resource and time constraints. In these cases, the developer must adopt the mindset of a tester – to look consciously for defects – for testing to be effective.  
  
The primary advantages of self-review include early defect detection and prevention and cost effectiveness.

However, higher levels of independence in testing produce more effective results, even though they may impact the schedule and project costs.

Question

As a Project Manager for a payroll application, you want to employ different levels of testing independence for different testing activities.  
  
Rank the levels of testing independence from the lowest to the highest.

**Options:**

1. Ask Pete, who designed and worked on the payroll application, to review his module and report defects detected.
2. Acquire the help of Sandra, who is working on a related application, to test the payroll module created by Pete
3. Hand over one module of the payroll application to the testing team in your organization that tests all completed software products of your organization
4. Hire certified testers from outside your organization to run tests on the payroll application

Answer

**Correct answer(s):**

**Ask Pete, who designed and worked on the payroll application, to review his module and report defects detected. is ranked**

The first level of testing independence is applied when the developer who created the software is employed to test the product. This level can be applied for cursory reviews.

**Acquire the help of Sandra, who is working on a related application, to test the payroll module created by Pete is ranked**

When a peer within the team is involved in testing a software product, then second-level independence is applied. This level of testing independence is higher than the level of self-reviews.

**Hand over one module of the payroll application to the testing team in your organization that tests all completed software products of your organization is ranked**

The third level of independence is applied when a member of a different functional group, such as the QA team in your organization, is involved in the testing process.

**Hire certified testers from outside your organization to run tests on the payroll application is ranked**

The highest level of independence is applied when a tester from a different organization or company is employed to test the software product developed in your organization.

**2. The role of communication in testing**

Although processes and procedures are critical to the success of software testing, an overlooked but equally important component affecting testing is communication.  
  
Because software testing includes the participation of multiple teams – internal testers, development team, stakeholders, and external testers – these teams must communicate with each other as seamlessly as possible. Otherwise, misunderstanding and misinterpretation can undermine the test objective.

Suppose your organization requires test activities at every stage of the process. The test results at every stage should be communicated to the entire team, because the tester may change for every round of testing.  
  
For example, a developer tests the software design model during the design phase and an integrator checks for integration defects later in the development stage. Results of these tests need to be communicated to the next level of testers and the final testing team so that they are aware of what occurred in the past and how it was resolved. It also helps them to look for similar defects while testing.

Not only is communicating initial test activities to the testing team important, but how the testing team communicates its test results to the development team is equally critical. A tester's mindset is to look for defects in a product, and when they find them; their role in the process is justified. This can affect the tone used when they log defects.  
  
A harsh tone can make a developer defensive. The developer may feel that the defect log pinpoints faults, which in turn reflects on the developer's skills and competency. This may lead him to disregard defects reported. The validity of the defect is lost and the purpose of testing is defeated. Effective inter-team communication can help avoid these issues.

For communication among team members during the test process to be effective, it is important to communicate

**defects and incidents neutrally**

Chances of test results being taken positively by the development team would increase if you communicate the defects and incidents in an objective manner. This means that the defects should be recorded without attributing the cause to the person who worked on it or undermining his effort. You should provide objective information on the defects so that the developer understands that you're not finding fault with him. For example, if the developer of the software has made a recurring error, simply explain the defect and discuss your observations of the incident. You can also record anything that you felt worked particularly well in the software.

**how the end user benefits**

For successful software development and testing, it helps to keep in mind the requirements and the benefits to the end user. Communicating how the defects you observe can impact end-user experience, provides objectivity to the test process. You should also emphasize how the defect prevents the product from meeting its stated objective. For example, you are evaluating a GUI-based web application that requires high network bandwidth usage. As a tester, you can highlight this to the development team and explain to them how this may hinder the end user experience.

**the need for collaboration**

The test process benefits from team effort, with the test and development teams collaborating effectively. In addition to achieving end-user satisfaction, working in a collaborating team helps you detect and fix defects, with multiple perspectives. Collaboration requires you to share your observations and suggestions with your team and receive feedback. For example, you've identified a new defect in a piece of software, but you are unsure of the scope of the defect. Instead of logging with the information you have, you could collaborate with the developer to get more information. As well as improving the information logged, this could help the developer feel more involved and invested in the testing process.

As a tester, you should remember and understand the objectives and the stakeholder requirements defined for the testing process. The project and the test process are driven by these factors, and test plans are based on them. Understanding these factors helps you adopt the right perspective while testing.  
  
For example, if testers understand the objectives, they can set test parameters and the exit criteria, and rate the performance of the product according to these details. Otherwise, testing can be flawed, with noncritical defects possibly emphasized over others. In such cases, the developer may reject the defects logged against the software, hindering the overall test process.  
  
Testers must be able to identify what is critical and what is not while reporting defects. This ensures that the development team does not spend time and money in closing defects that are not critical to delivery.

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use.   
  
Recognizing the ACM and IEEE code of ethics for engineers, the ISTQB® states the following code of ethics:

**public**

Certified software testers shall act consistently with the public interest.

**client and employer**

Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

**product**

Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible.

**judgment**

Certified software testers shall maintain integrity and independence in their professional judgment.

**management**

Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing.

**profession**

Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest.

**colleagues**

Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers.

**self**

Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Question

You implemented a test suite and found some incidents and defects. Now you want the programmer who developed the application to fix them. How would you communicate your findings to the programmer?

**Options:**

1. Record incidents in a neutral and factual manner
2. Record your surprise on how such defects were allowed to creep in
3. Discuss how closing the defect helps the end product
4. Direct that all incidents must be closed without any delay

Answer

***Option 1:****Correct. As a tester, you need to create neutral and factual incident reports that clearly and constructively discuss the defects, without insulting or blaming the developer.*

***Option 2:****Incorrect. When you log errors against a programmer, your reports should be written in an objective tone devoid of personal remarks. Expressing surprise in incident reports detracts from objective testing.*

***Option 3:****Correct. As a tester, you need to explain to the developer how your observations would help improve the end product. This helps the developer recognize the testing process as an effort to improve the product.*

***Option 4:****Incorrect. Once you report the defects against the programmer and explain your suggestions, the programmer takes over. Insisting on a particular schedule is not a tester's domain.*

**Correct answer(s):**

1. Record incidents in a neutral and factual manner  
3. Discuss how closing the defect helps the end product

**3. Summary**

The process of software testing is influenced by psychological factors that determine the success of the testing activity. These factors help you understand the differences between the mindsets of a developer and a tester. Generally, objective testing – testing carried out by someone other than the developer – is preferred because it supports the separation of responsibilities across the various teams involved in testing. This separation of responsibilities is called independence.  
  
There are four basic levels of independence, based on the degree of separated responsibilities. These levels include tests executed by the programmer himself, a peer programmer, a tester from the same organization, or a test team from another organization.  
  
Communication across teams also affects test results. Test results should be communicated in an impersonal, nonjudgmental manner so that they are taken as constructive critiques.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

Contrasting Software Testers and Developers

Learning Objective

*After completing this topic, you should be able to*

* *recognize the different mindsets of a tester and developer in a test situation*

**1. The mindset of testers and developers**

The success of software testing is dependent on effective communication across different teams that participate in this activity, including testers and developers.

As a software tester, you need to interact with the programmers or developers, who build the software applications and services being tested. This interaction drives the entire process, helping achieve the project goals and objectives. Project objectives drive the test plans, and to understand these objectives thoroughly, developers and testers need to interact with one another seamlessly.

The interaction between the testers and the developers are influenced by numerous factors – psychological differences, roles and responsibilities in the testing process, and more.  
  
The roles and responsibilities of testers and developers impact their differing attitudes toward testing.

For example, the basic responsibility of a developer is to code and to create a software application based on customer requirements. Even if the developer performs a self-review during development to locate and fix bugs, she is likely to undertake testing from the perspective that the product is defect free. A tester, on the other hand, is not expected to create or code software, and may not factor in the effort the developer puts in. She may only focus on testing the software on the assumption that it does not meet customer requirements and so consciously looks for defects in the application.

There are basic differences in the mindset of testers and that of developers.

**Tester**

A tester tests the product for defects, keeping in mind the complete picture of the development life cycle and how different systems in the application are interconnected. Consequently, he or she is most likely to locate defects in the product. Because the tester didn't create the product, he or she is more objective and more likely to find defects.

**Developer**

A developer is usually an expert in a specific technical domain. Developers understand how a specific development model works and what functionality provides a better user experience of the product.  
  
However, because they delve deeply in their domains, they may lose perspective of the complete picture. They may discount some defects reported if they feel that the tester isn't competent enough to understand the development model or the technology used. Being personally involved in the development may make them less likely to accept defects reported objectively.

Testers should display competence by thoroughly understanding the objectives and requirements of the application. In doing so, they should get the developer's perspective so they know what is critical and valid. This understanding is reflected in the test cases and in the nature of defects reported. This competence helps convince the developer to fix the defects.   
  
In addition, testers should demonstrate maturity and integrity in communicating defects. They must avoid personal and judgmental communication. At the same time, they must maintain integrity and resist pressure from any stakeholder to allow the product to pass without thorough testing.

The mindset of testers is to look at products being tested as flawed, and they approach testing as an activity that uncovers these flaws. So they design test suites to detect defects instead of designing test cases that only check if something works. This results in more defects being uncovered.

In the same test situation, a developer is likely to fit the defects reported within the established development models, and in cases where they don't fit, disregard them. The developer aims to create a working solution, not to look for intermittent bugs that arise in some other element of the application.

The differing mindsets of the testers and developers can lead to misunderstandings and communication gaps when the two interact. There are ways that testers and developers can bridge this gap:

* understand and appreciate each other's roles and responsibilities in the testing process
* communicate and coordinate with one another throughout the development and testing life-cycle
* initiate conversations with the programmers, as testers, and not wait for them to start inquiring about defect reports
* share plans, drafts, schedule, design documents, and prototypes to stay updated of the process
* provide fair feedback to the developers, appreciating their good work and reporting defects objectively

A tester should be able to discriminate between the critical factors that need immediate attention and those that are minor. This helps him log faults that are valid and require immediate attention. He should also maintain logs and incident and test summary reports as evidence for test defects. This helps him substantiate defects when questioned by the developer.  
  
Although reporting defects is a key responsibility, the tester must also communicate the fault appropriately to the developer so that it can be fixed. The tester should avoid offending the developer and present the defect report in a manner that makes the developer understand that the aim is to improve the end product.

Question

What mindset should a potential tester adopt while analyzing a test case?

**Options:**

1. Understand the need to observe, identify, and record evidence of the defects
2. Recognize and fit defects found within the development models used
3. Be open to appreciate good work in the application in addition to reporting defects

Answer

***Option 1:****Correct. A potential tester should always back the defects identified with evidence, such as test logs and test summary reports. Doing so provides a clear perspective of the defects and the conditions they occur in, and allows developers to validate and fix the defect.*

***Option 2:****Incorrect. Developers work based on established development models and usually try to fit defects within these models when they're reported. Testers only need to understand the basics of these models so that defects can be communicated effectively.*

***Option 3:****Correct. A good tester always provides fair feedback, which includes positive and constructive feedback, using an incident report.*

**Correct answer(s):**

1. Understand the need to observe, identify, and record evidence of the defects  
3. Be open to appreciate good work in the application in addition to reporting defects

**2. Summary**

The success of software testing is dependent on the effective interaction between testers and developers.  
  
To improve this interaction, you should first understand the differences in the mindsets of a developer and a tester. These mindsets are dependent on their roles and responsibilities. Testers possess an overview of the entire development life cycle, and the developers are experts in a specific technical domain. Developers code and create software, and testers review these software products and identify the defects within.  
  
The responsibility of a tester does not end with reporting the defects. He should communicate the defects politely to the developers and offer suggestions to fix the defects. He should provide fair and objective feedback. This helps bring testers and developers together and creates a unified team.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

Organizing Testers and the Software Test Process

Learning Objectives

*After completing this topic, you should be able to*

* *organize a test team*
* *organize test activities*

**1. Exercise overview**

In this exercise, you're required to set up a test team and organize activities that constitute the software testing life cycle.

This involves the following tasks:

* setting up a test team
* organizing test activities

**2. Setting up a test team**

You are a test manager in an organization. A development team within your organization has created a payroll administration web site for the company, which will help payroll executives manage the payroll account of all employees. The site also helps employees view their salary credits, view and print pay slips, and claim monthly reimbursements.  
  
As a test manager, you want to test and evaluate the functionality of the payroll intranet. To begin, you want to set up a core team of potential testers.

Question

You want the team you set up for testing the payroll application to have a degree of independence and objectivity. Due to budgetary limitations, you cannot opt for the highest level of independence.   
  
Who would you assign as testers considering the level of independence required?

**Options:**

1. The developer who worked on the application
2. A peer of the developer from within the team
3. A testing team from within the organization
4. Testers from a certified testing organization

Answer

***Option 1:****Incorrect. This is the lowest level of testing independence and should be applied for cursory reviews of the product, not for achieving higher degrees of testing independence.*

***Option 2:****Correct. The second level of testing independence is achieved when peer reviews are implemented in testing a software product. This level is higher than self-reviews and provides a fresh perspective lacking in self-reviews. In the absence of a testing team, this is an option to consider.*

***Option 3:****Correct. For a higher level of testing independence, you can employ an internal tester from a different functional group or a testing team to test the product. This is the preferred level of testing independence in most organizations.*

***Option 4:****Incorrect. Although certified testers from organizations that specialize in testing provide the highest level of independence, it is expensive. With budgetary limitations, this isn't a feasible option.*

**Correct answer(s):**

2. A peer of the developer from within the team  
3. A testing team from within the organization

Question

You decide to use testers from a different functional group to test the product to get the level of independence required. You want to ensure that the testers on your team display the right mindset.  
  
Identify the mindset that a potential tester should adopt while analyzing a test case.

**Options:**

1. Upgrade their skills to gain expertise over the specific domain in the development model being tested
2. Provide evidence in support of defects observed at all times
3. Discriminate between critical defects that require immediate attention and those that are minor
4. Thoroughly understand the objectives and requirements of the application under test

Answer

***Option 1:****Incorrect. Obtaining the working knowledge of a specific domain in the development model is the responsibility of a developer, who is considered a specialist in that domain. A tester needs only an overall idea of the entire test case.*

***Option 2:****Correct. A potential tester should always back the defects identified with evidence, such as test logs and test summary reports. This helps developers validate and quickly fix the defect.*

***Option 3:****Correct. Testers should be able to differentiate between the two to help save both time and cost of the overall testing process.*

***Option 4:****Correct. Understanding the objectives and the requirements of the application helps testers to design and implement test suites that help them detect defects.*

**Correct answer(s):**

2. Provide evidence in support of defects observed at all times  
3. Discriminate between critical defects that require immediate attention and those that are minor  
4. Thoroughly understand the objectives and requirements of the application under test

Question

You also want the test team to focus on communication. Suppose a test engineer in the team finds a simple but valid defect in the login functionality of the payroll application. This defect is to be communicated to the senior programmer who designed and developed the program. How would you want the test engineer to communicate this defect to the programmer?

**Options:**

1. Create a strongly worded incident report escalating how such a basic but major defect could occur
2. Explain the defect with observations and defect history in a factual incident report
3. Communicate the defect in a neutral tone with remarks on how fixing it helps the end user
4. Inform the programmer that the defect is assigned to someone else with experience

Answer

***Option 1:****Incorrect. A good tester creates neutral and factual incident reports, and contributes as a team rather than blame anyone for defects found. A strongly worded incident report can cause the programmer to feel that he or she is being judged.*

***Option 2:****Correct. The incident reports that a tester creates should be neutral and objective. For this, the defects should be recorded without attributing the cause to the person who worked on it or undermining the effort of that person.*

***Option 3:****Correct. Testers should always communicate their opinions – subjective and objective – in a neutral manner. This emphasizes to the developer that the tester's only interest is to ensure that the product meets the stated objective and meets end-user expectations. This tone also makes it easier for developers to accept the feedback positively.*

***Option 4:****Incorrect. A potential tester should never assume that a defect cannot be fixed by the programmer who worked on it. The programmer has the domain knowledge and is skilled enough to fix a major defect in design.*

**Correct answer(s):**

2. Explain the defect with observations and defect history in a factual incident report  
3. Communicate the defect in a neutral tone with remarks on how fixing it helps the end user

**3. Organizing test activities**

Now that you've set up a core testing team, you want to identify the order of test activities that need to be performed to carry out the test process.

Question

You want the focus of the test team to be client privacy and security for the payroll application. Sequence the tasks in the order your team will perform them during the software testing process for this payroll site.

**Options:**

1. Identify exit criteria based on client privacy and security objectives and create test plans
2. Specify as a test condition that only authorized employees with valid user IDs can enter the site
3. Create a test suite for the payroll administration site with test procedures that check if individual logins of employees could be circumvented
4. Check the logs of the actual test outcome to check if the exit criteria on employee privacy is met

Answer

**Correct answer(s):**

**Identify exit criteria based on client privacy and security objectives and create test plans is ranked**

Understanding the goals of the project and identifying the objectives and risks belong to the planning phase of the testing process. During this phase, the testers can also determine the kinds of tests to run.

**Specify as a test condition that only authorized employees with valid user IDs can enter the site is ranked**

During the test analysis phase, the test documents – such as requirements and design specifications – are analysed.

**Create a test suite for the payroll administration site with test procedures that check if individual logins of employees could be circumvented is ranked**

During the test implementation and execution phase, the testers on your team will convert the test conditions that were determined during test design into test cases and test suites. For the payroll site, these test suites would focus on the security features of the site.

**Check the logs of the actual test outcome to check if the exit criteria on employee privacy is met is ranked**

Checking the logs of the actual test outcome against the exit criteria is carried out during the exit criteria evaluation phase. During this phase, your testers would analyze if the exit criteria for the privacy of the site is adequate, or if more tests are required.

Question

After test suites are executed and test results evaluated, you now want your team to close the test process. What test activities should the team perform to close the test process?

**Options:**

1. Archive testware for maintenance testers
2. Analyze lessons learned for future test projects
3. Recommend additional tests to resolve open defects in the payroll processing site
4. Check the test logs against the exit criteria set for the payroll application

Answer

***Option 1:****Correct. Testers archive testware so that maintenance testers can reuse the same components during maintenance testing and regression testing – because it is time- and cost-effective.*

***Option 2:****Correct. In the project evaluation report, testers identify and assess the lessons learned from this test process, which would help them improve the test process as well as the development life cycle in the future.*

***Option 3:****Incorrect. A tester determines whether additional tests are required during the exit criteria evaluation phase, not in the test closure phase.*

***Option 4:****Incorrect. Checking the test logs against the exit criteria set for the payroll application occurs during the exit criteria evaluation phase. This test activity is not part of the test closure phase.*

**Correct answer(s):**

1. Archive testware for maintenance testers  
2. Analyze lessons learned for future test projects

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_sftf_a01_it_enus/output/html/course_transcript.html#top)

© 2017 Skillsoft Ireland Limited