

**Chapter  
1**

**GETTING  
STARTED**

*Get on the  
Fast Track!*



TM

**SYS-ED/  
Computer  
Education  
Techniques, Inc.**

TM

## **Objectives**

You will learn:

- Purpose of testing.
- Organization of testing effort.
- Requirements-based testing.
- Developing a test plan.
- Test plan approaches.
- Waterfall approach.
- Evolutionary approach.
- Unit testing.
- Integration testing.
- Regression testing.
- Building a library.

---

## 1 Purpose of Testing

Testing measures the quality of the software being developed. This view assumes that there are defects in the software waiting to be discovered.

Several factors contribute to the importance of testing.

- Reducing the cost of development

Common estimates indicate that a problem that goes undetected and unfixed until a program is actually in operation can be 40 – 100 times more expensive to resolve than resolving the problem early in the development cycle.

- Ensuring that the application behaves as intended

Unpredictability is the least desirable consequence of using an application.

- Reducing the total cost of ownership

Applications that look and behave as shown in the documentation, will require fewer hours of training and less product support.

- Developing customer loyalty.

Effective testing translates into a better quality product.

---

## **2 Testing Implementation**

The earlier in the development cycle that testing becomes part of the effort the better. Planning is crucial to a successful testing effort, in part because it has a great deal to do with setting expectations. Planning also ensures tests are not forgotten or repeated unless necessary for regression testing.

---

### **2.1 Requirements-Based Testing**

The requirements section of the software provides the basis for all testing on the product.

Specification writers will need to maintain the following standards when presenting requirements:

- All requirements should be unambiguous and interpretable only one way.
- All requirements must be testable in a way that ensures the program complies.
- All requirements should be binding because customers demand them.

Test cases need to be designing as the specification is being written. Each specification needs to be assessed in terms of how well it supports the development of test cases.

---

### **2.2 Developing a Test Plan**

The test plan outlines the entire testing process and includes the individual test cases.

In order to develop a solid test plan, the program will need to be systematically explored to ensure coverage is thorough. A formal test plan establishes a testing process that does not depend upon accidental, random testing.

---

### **3 Test Plan Approaches**

Two common test plan approaches to testing are:

- waterfall approach.
- evolutionary approach.

---

#### **3.1 Waterfall Approach**

The waterfall approach is descends directly from the development team in which each person works in phases, from requirements analysis to various types of design and specification, to coding, final testing, and release.

A significant disadvantage of this approach is that it eliminates the opportunity for testing to identify problems early in the process; therefore, it is best used only on small projects of limited complexity.

---

#### **3.2 Evolutionary Approach**

An alternative is the evolutionary approach in which a modular piece or unit of an application an application is developed. It is then tested, fixed to reasonable degree of completeness, and then another small piece is added which then adds functionality. The two units are then tested as an integrated component, increasing the complexity gradually.

Advantages to this approach are:

- It is cheaper to reappraise requirements and refining the design, as the application is understood better.
- A working, useful product is being constantly delivered.
- Instead of one large test plan, the project can can start with small, modular pieces of what will become part of the large, final test plan.
- New sections can be added to the test plan and in-depth testing can be performed in new areas.

---

## **4 Optimization**

Optimization is the process by which bottlenecks are identified and removed by tuning the software, the hardware, or both.

The optimization process consists of four key phases:

- collection
- analysis
- configuration
- testing

Without baseline performance data, it will not be possible to determine if the modifications helped or hindered the application.

---

## **5 Unit Testing**

The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as expected.

Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use.

The most common approach to unit testing requires drivers and stubs to be written. The driver simulates a calling unit and the stub simulates a called unit. It allows for automation of the testing process, reduces difficulties of discovering errors contained in more complex pieces of the application, and test coverage is often enhanced because attention is given to each unit.

Finding the error or errors in the in an integrated module is much more complicated than first isolating the units, testing each, then integrating them and testing the whole.

Drivers and stubs can be reused in order that the constant changes that occur during the development cycle can be retested frequently without writing large amounts of additional test code. This reduces the cost of writing the drivers and stubs on a per-use basis and the cost of retesting is better controlled.

---

## 6 Integration Testing

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. The basic premise is to test combinations of pieces and eventually expand the process to test a module with those of other groups. Eventually all the modules making up a process are tested together.

Integration testing identifies problems that occur when units are combined.

There are three common strategies to Integration testing:

Top-down Approach	Requires the highest-level modules be tested and integrated first. This allows high-level logic and data flow to be tested early in the process and it tends to minimize the need for drivers.
Bottom-down Approach	Requires the lowest-level units be tested and integrated first. These units are frequently referred to as utility modules. By using this approach, utility modules are tested early in the development process and the need for stubs is minimized.
Umbrella Approach	Requires testing along functional data and control-flow paths. The primary advantage of this approach is the degree of support for early release of limited functionality.

---

## **7 Regression Testing**

A regression testing needs to be performed whenever an implementation within a program is modified. This can be done by rerunning existing tests against the modified code in order to determine whether the changes break anything that worked prior to the change and by writing new tests where necessary.

Strategies and factors to consider during this process include:

- Test fixed bugs promptly; programmer might have handled the symptoms but not have gotten to the underlying cause.
- Watch for side effects of fixes; the bug itself might be fixed but the fix might create other bugs.
- Write a regression test for each bug fixed.
- If two or more tests are similar, determine which is less effective and get rid of it.
- Identify tests that the program consistently passes and archive them.
- Focus on functional issues, not those related to design.
- Make changes (small and large) to data and find any resulting corruption.
- Trace the effects of the changes on program memory.

---

### **7.1 Building a Library**

The most effective approach to regression testing is based on developing a library of tests made up of a standard battery of test cases that can be run every time a new version of the program is built. The most difficult aspect involved in building a library of test cases is determining which test cases to include.

The regression test library should be reviewed periodically to eliminate redundant or unnecessary tests. The general recommendation is that it should be done about every third testing cycle.