Effective Testing Strategies

By **Ellen Whitney**

Share

Ellen Whitney

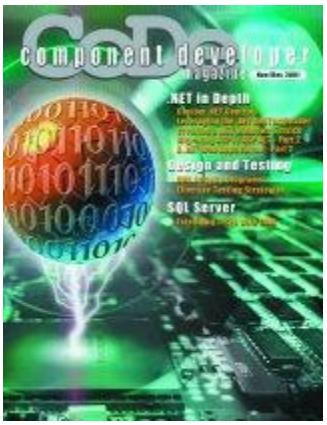
Ellen Whitney is the Vice President and CFO of EPS Software Corporation. She is also Managing Editor of CODE Magazine.

Ellen has an extensive backround as a senior developer specializing in the design and implementation of object-oriented software systems. Her thorough knowledge of the development process has benefited many clients as well as having made her a lecturer and international author.

In her current role she works extensively with the Sales and Marketing Departments and is responsible for setting the overall goals and direction of the company.

In her spare time, Ellen enjoys watercolor, acrylic and oil painting, as well as hiking and both indoor and outdoor cycling.

This article was published in:



This article was filed under:

- Tips
- Web Services

Do you test your software before you release it?

Of course you do, but are you testing it as effectively as you could be? When do you start testing? How do you know when you have tested it enough? Who does the testing? This article will explore the various strategies of creating a comprehensive testing process for your software development project.

Introduction

In many projects, developers wait until the middle or the end of the software development process to worry about thoroughly testing their program. In other words, developers often test portions of their code as they are programming, but wait until the software is nearing completion to examine the software as a whole to determine if it is what the client wants and if it is functioning properly. In this article we'll explore other options ? perhaps there is a better way! We'll take a look at who should do the testing, how testing should be performed and when it should be done. After taking a look at some other possibilities, I think that you'll find that there are many benefits to starting the testing process much earlier in the software development process instead of waiting until the middle or end.

It's extremely important that any errors that are discovered are documented including describing the steps to reproduce them. This makes it possible for the developer to diagnose and fix the offending code. The various tests that are performed should be thoroughly documented so that the tests can be run multiple times and their results can be analyzed.

In this article we will only be exploring software testing ? that is finding various defects and omissions in the way in which the software functions. We will not be discussing debugging ? the process of finding where these defects occur or how to fix them.

Who should do the testing?

Who should be responsible for doing the testing on a software project? In many companies the developer is also responsible for doing the majority of the testing. But wait a minute! Let's examine the two tasks involved. Development and testing are in some ways fundamentally opposite. Developers are responsible for creating a product while testers are trying to expose its faults or break it. In some ways, being a skilled tester is more difficult than being a skilled developer since testing requires not only a good understanding of the system being developed, but also the ability to anticipate likely points of failure. Therefore, since each of these tasks is quite different, different types of people are naturally better at each task. A good tester has the following qualities:

Skeptical they want proof that the software works as it's supposed to.
Objective they make no positive or negative assumptions.
Thorough they are committed to testing every area and possibility that has been determined necessary.
Systematic and organized they can efficiently create reproducible scenarios to offer as proof of failure.
Another challenge with developers performing the bulk of the testing is subconsciously, developers do not want their code to fail. They do not want to discover errors in their code. Therefore it's extremely difficult for a developer to objectively and thoroughly test their own code.

Ideally there should be separate dedicated software testers in addition to the developer staff. If the size of the project prohibits having a person dedicated solely to testing, an alternative solution is to have developers test each other's code instead of only testing their own. **Figure 1** shows a possible hierarchy for a software project that has a testing

team independent of the development team. Of course the exact structure will depend on the size of the development project.

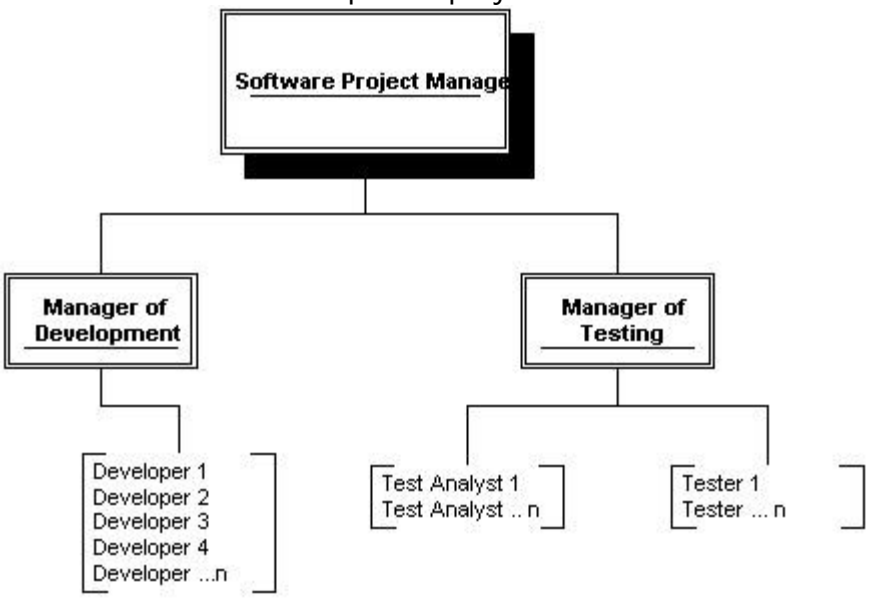


Figure 1 Software Project Organizational Chart

The Testing Process

So how does testing fit into the Software Development Process? In many projects, testing begins after implementation when the developer tests some of the code that they've written. Once the developers are satisfied with their code, the program is deployed and the process moves on to alpha or beta testing (**Figure 2**). But is this really enough? Is this the best way? Does this effectively satisfy all our requirements for creating solid software the most cost effective way possible? As we will explore, a better way is to integrate testing into the entire development process (**Figure 3**). In **Figure 3** you will see for each task that the developer performs, there are also testing tasks that are performed concurrently.

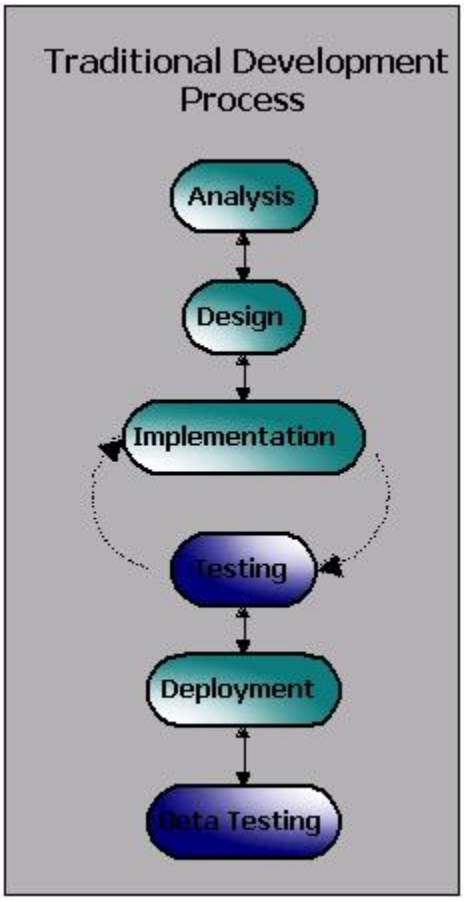


Figure 2 A common, but perhaps inadequate view of where testing fits into the Software Development Process

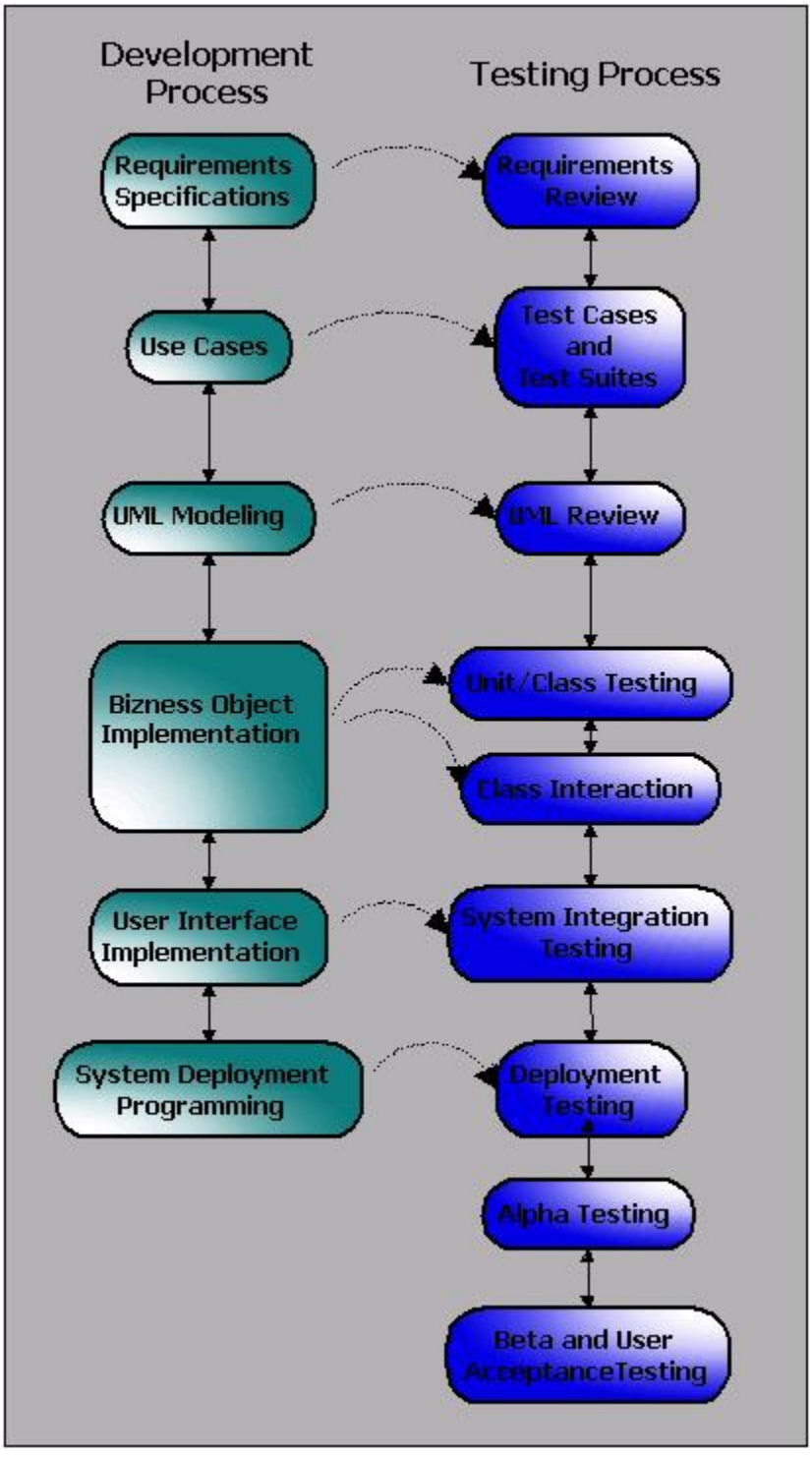


Figure 3 The integrated Software Development and Testing Process

The Testing Process, like the Software Development Process varies somewhat based on the type, size and complexity of the project. The goal of developing the best possible testing process is to test as thoroughly as possible to insure that there are no omissions in the client specification and to eliminate as many of the defects as possible. However,

testing comes at a cost. How much testing should you do? When are you done? The answers to these questions will depend on the type of software you are developing. The amount of testing effort should be sufficient to guarantee the functionality of the product with respect to the importance of the product or routine being developed. In other words, what is the risk of having an unfound defect in that portion of the program going to cost? Are the risks of failure a life, an injury, a significant loss in revenue or merely an annoyance or inconvenience to the user? Considering these risks and working to mitigate them is what Risk Management is all about. In addition to Risk Management being an integral part of the Software Development Process, these are also some of the factors that must be weighed in order to determine when you have tested "enough". In many projects, testing is determined to be completed based on a fixed "due" date or a fixed monetary amount. This is definitely not the best way. As we discuss each of the testing processes, we'll also take a look at when the testing process for each task is considered "complete".

Requirements Review

The first step in the development process is gathering requirements from the domain expert and/or stakeholders to determine a list of specifications for the system. (For more information about gathering requirements, see the article I wrote for *Component Developer Magazine 2001 - Issue 2*.) It is recommended that the primary tester or testers also be involved to some extent in this discovery phase so that they fully understand the scope of the system. The degree in which the testers are involved at this point depends on the complexity of the system. If the project is one that most people are familiar with, having the testers involved at this point is not nearly as critical as a project that is highly technical or unfamiliar.

Once requirements lists have been created, the Test Analyst should act as a reviewer to review the lists of requirements to be sure everything that has been discussed with the client has been properly documented. In the process of reviewing the requirements, careful attention should be given to the clarity and thoroughness. Too many times the requirements are ambiguous and the reader (the Test Analyst or Developer) assumes that he just doesn't completely understand the domain. However, ambiguities should be completely resolved in this phase since it may be an indication that there is a conflict among the domain experts. Many times ambiguities actually allow the domain experts with different goals or opinions to all be "right" and satisfied with the specification because it is open to individual interpretation. These issues however must be resolved as soon as possible since the requirements are the foundation on which the entire project is built. The system specifications are likely to become blurred during the process of development, they are never going to become clearer than in the requirements.

Additionally, changes at this stage of the process are relatively cheap, so it's best to resolve them before any actual code has been written.

Once all the Developers, Test Analysts, Domain Experts and Stakeholders are comfortable with the entire list of requirements, the Requirements Review is considered complete.

Test Cases and Test Suites

The next step in the Software Development Process involves producing Use Cases. Use Cases integrate the requirements into a comprehensive package that describes the interaction of the user with the system. Use Cases are text documents, written in plain English, which describe a scenario in which an "Actor" (the user or another mechanical device or system) interacts with the "System" (the software we are building). **Figure 4** shows a portion of the Scenario section of a sample Use Case. For more information about writing Use Cases, see the article I wrote for *Component Developer Magazine 2001 - Issue 2*. The first duty that the Test Analyst should perform is to review these Use Cases to make sure that they have documented each scenario completely and unambiguously.

Scenario:

	Action (Stimulus)	Software Reaction
1.	Customer indicates that they would like to purchase tickets.	System requests the following data from the Customer: • Movie • Theater • Time • Number of tickets For the movie, theater and time, the system shows the Customer a list of valid choices. For more information, see **Use Case 003: Managing Pick Lists**
2.	Customer fills out data.	The System verifies that the above 4 items have been filled out. If any data is missing, the System warns the Customer and the scenario continues with Software Reaction #1. If all data has been entered, the System asks the Customer to verify their purchase. If the Customer indicates they would like to cancel the scenario ends here. If the Customer indicates that their order is not correct, the scenario continues with Software Reaction #1.

Figure 4 A sample Use Case

Secondly, using the Use Cases as reference, the Test Analyst should produce Test Cases. Test Cases are arguably the most important component of the testing process. To create Test Cases from Use Cases, each line of the scenario is made more specific by assigning exact values or user actions. These values describe an input to the section of the software

being tested, and also the expected output. The inputs and outputs often are not merely simple data values, but can be extremely complex. For example, input and output could involve a series of user interactions, feedback to the user such as a message box or a sound, values being saved to a database or even a change to the program state that affects future reactions of the software. **Figure 5** shows some possible Test Cases for the Use Case shown in **Figure 4**.

Sample Test Cases for Scenario 1 and 2:

	Test Scenario	Values	Software Reaction
1.	The movie is blank. *(Movie can not have other invalid data since it's a pick list)*	Movie = blank	PASSED - System displays warning "Please fill in Movie title"
2.	The theater is blank. *(Theater can not have other invalid data since it's a pick list)*	Theater = blank	PASSED - System displays warning "Please fill in Theater name"
3.	The time is blank. *(Time can not have other invalid data since it's a pick list)*	Time = blank	PASSED - System displays warning "Please fill in time"
4.	The number of tickets is blank.	Number of Tickets = blank	PASSED - System displays warning "Please fill in number of tickets"
5.	The number of tickets is negative.	Number of Tickets = -3	PASSED - System displays warning "Number of tickets can not be negative"
6.	The number of tickets is too large (greater than 20).	Number of Tickets = 21	FAILED – System did not display warning
7.	Test using only the keyboard. Is the tab order between the fields right? Can all fields be accessed and manipulated without a mouse?	Tabbed between all fields.	PASSED – All fields are in the proper order and can be accessed via the keyboard.
8.	All fields are filled out with valid data. *(Be sure to also test valid cases!)*	Movie = "Star Trek III" Theater = "Stars Theater" Time = "7:30 PM" Number of Tickets = 2	PASSED – System accepts data

Figure 5 Test Cases are the most important element of the Testing Process

Each Use Case should be scored according to how crucial that task is to the overall system. In other words, what is the degree of risk associated with the Use Case performing incorrectly? A separate score should be given for the frequency in which the Use Case occurs. It is a combination of this risk and frequency that allows the Testers to prioritize the Test Cases.

Each Use Case will almost always result in several Test Cases. The exact number of Test Cases required to adequately cover each line of the Use Case scenario will depend on the defined software reaction and the number of alternative paths. When the Test Analyst is writing the Test Cases, it is extremely important that both valid and invalid values are tested. A good Test Analyst will have the experience to guess values that are likely to

produce errors. Test Cases are normally considered "black box testing". This means that the Test Cases are written without knowing exactly how the code has been implemented. All that is known is the expected output for given input values.

Test Cases are normally organized into test suites or test scripts. Test suites are a logical grouping of Test Cases based on some commonality between them. For example a test suite might contain tests dealing with system performance, a certain subsection of the program or a certain series of reports. Test suites permit the tester to logically organize the individual Test Cases allowing easier handling, reporting and management.

UML Review

The next step in the Development Process is for UML models to be built. These models are used to design the structure of the system to be built. For more information about the UML, see the article I wrote for *Component Developer Magazine 2001 - Issue 3*. This step will require the Test Analyst to have experience in UML modeling. During this step, the Test Analyst should review the models to be sure that all the information from the Use Cases has been accurately and thoroughly reflected in the models.

Unit/Class Testing and Class Interaction Testing

Following the UML modeling, the developer will begin writing actual code and to start to building classes. At this point the initial testing should be performed by the Developer. There are several testing methodologies that in my opinion should be strictly adhered to by each Developer. First, the Developer should step through every single line of code while watching it run in a debugger. Even though most people would agree that every software program will have bugs; syntax errors are completely unacceptable. If each Developer runs every single line of code he writes, syntax errors would be obsolete to the end user. Many programming languages also have internal tools that allow the developer to identify any lines of code that have not been run during testing. Developers should use these tools to become more efficient at testing.

Secondly, I am a strong believer in having an automated testing script to efficiently complete this step in the Testing Process. This doesn't need to be very complicated, but must provide a method in which the developer can write and save test scenarios. Using this methodology, the developer can be assured that in the process of implementing new code no old classes have been broken. For this portion of testing, a short script is required

that will run thru test scenarios stored in a database. Each record should have the name of the object to be created, the name of the method to be run and the parameters to be passed. It also must have a place for recording the actual vs. the expected results. In the script that I wrote, I also included a flag field that determines which tests should be run during any given test session. This saves time running the script since not every test is run each time the test script is activated. Thus for each object I create, I create several tests that demonstrate both passing and failing values for each method of the object. I thoroughly test each object as I create it and then save the individual tests for future testing. So in the future, when I create a new class or need to change the code in a base class, I can easily just run through my entire set of tests to insure that the new code has not broken anything written previously. This methodology has proven invaluable to me and was well worth the few hours I spent creating my test program code. After the developer is satisfied that his code is solid, I highly recommend that a tester or at least another developer test each class.

It is also extremely important that the interaction between the classes is tested. To adequately test each class, it's important to consider who the internal user of the class will be and how it will be called, as well as how the classes will interact with each other. Testing in this phase is usually both "structural" or "white box testing", in which how the code is constructed is taken into consideration and designed into the tests, as well as "black box testing", where only the input and output values are examined.

So in this step of the testing process, when can testing be considered complete? It would be great to be able to test every line of code, every path, every interaction between objects and every input and output value of every single method. However, this is normally not feasible. This is where it's important to consider the underlying design and the language in which the software is programmed in. With object oriented systems, once a base class has been thoroughly tested often other classes that inherit from this base class will be much faster and easier to test. Learn to look for the exceptions and test them. Also, each language lends itself to the possibility of containing certain kinds of errors and by default eliminates others. For example, if the language you are using is strongly typed it will significantly reduce the number of interface errors that will occur because of mismatched parameters since the compiler will insure the proper type has been used. Take the structure of the programming language into consideration when determining the most important things to test for.

System Integration Testing

Finally, the Developer creates the User Interface. The type of testing that is done after this step is probably the one that people are most familiar with. In many Software Processes, this is the first time that testing really comes into play. However, as we have seen, that there is a huge amount to be tested prior to the creation of the UI!

When testing the UI, it's important to test various modes of operation. What happens if the user only uses the keyboard and doesn't use the mouse? What happens if the user doesn't follow the "predicted path" of operation? What happens if the user quits before they've completed the operation? Try to think like a new user versus an experienced one. Does the system accommodate both? It is this type of thinking that separates a good tester from a mediocre one.

Testers will use the Test Cases described above to know exactly what to test in this step. They will use the score based on the risks involved and the frequency of occurrence to prioritize their tests.

Depending on the client and the type of system being developed, it may be important to have the domain experts and/or actual users review the system at this point to get feedback about the look and feel of the UI. Even though careful attention has been paid to gathering good specifications, often other ideas will surface once the users can see the actual system running.

Deployment Testing

One important area that is often overlooked in testing is Deployment Testing. If the software is to be installed on many machines, it is extremely important to thoroughly test the installation process. Does it work on all operating systems? What are the minimum hardware requirements? Does the software co-exist peacefully on machines that have various other programs installed? Could some users have a previous version installed? Deployment is normally difficult to thoroughly test since there are so many variables. Thus it's extremely important to have a solid testing plan and the necessary hardware to be sure the program functions as advertised.

Alpha Testing

Once the basics of the system have been completed, it's a good idea to perform Alpha testing. With Alpha testing the Tester tests the overall system in a method mimicking the

way a real user would. During Alpha Testing, and then again later during Beta Testing, it's important that the Testers have a concise and standardized method of reporting any problems they encounter. This is where bug reports, or bug tracking programs are used (see **Useful Web Links sidebar**). The exact items that a bug report includes may vary slightly between projects; however **Figure 6** should give you an idea of the most important ones.

PROJECT BUG REPORT

Identifier: Unique name or bug number

Status: (New, Fixed, Inprocess, Completed)

Tested By: Tester's Name **Date:** Report date

Developer: Developer's Name

Preconditions: Any preconditions that must exist in order for bug to be found

Description: Brief description of bug

Exact Steps to Reproduce: Step-by-step list to reproduce bug

Expected Results: Description of what should have happened

Type of defect:

- ☐ Bug
- ☐ Design Flaw
- ☐ Enhancement Request
- ☐ Other _____

Severity:

- ☐ High
- ☐ Medium
- ☐ Low

Priority:

- ☐ High
- ☐ Medium
- ☐ Low

Notes: Any additional notes

Figure 6 A sample Bug Report

During Alpha Testing it is often a judgment call as to determine much testing is enough. Part of the answer lies in the number of problems that are found per hour of testing. Again the risks involved will help determine what an acceptable level of bugs is. However, the

best metrics can not take into consideration an extremely valuable resource ? "instinct". Before moving on to the next step, talk to the Testers and Developers. Ask them if there are any areas of the program that they don't feel completely comfortable with. Learn to trust people's instincts. If someone is uneasy about a portion of the program, it very well may be worth as second look.

Beta and User Acceptance Testing

Once the system has passed the Alpha Testing stage, it's time for the program to be tested by several real users. It is important that the users be a representative sampling of the actual users. In other words, not all the beta testers should be highly experienced, but there should be beta testers who represent the novices as well as the average user. These Beta Testers will need to be educated in how to report any problems they find since it's vital that they have a useful and efficient method to communicate back to the Developers. Often a similar if not identical bug reporting system used during Alpha Testing can also be used here.

Once the system has been released, there may be a period of User Acceptance Testing. This is the final phase of the testing process. Again, similar bug reports may be used. At some point you may wish for the client to sign off on each module saying that it is complete and functioning properly.

Summary

I hope that this article has given you a broader view of software testing and has given you some ideas for improving the testing process on your project. I hope that I've been able to convince you that there are many benefits to having a complete Testing Process in conjunction with the Software Development Process. If at all possible, the people that will be responsible for the testing should also be involved in the requirements and Use Case portion of the project since it is imperative that these testers have a thorough understanding of the requirements.

It's also important to be aware that testing requires resources and these resources require careful planning and good management. It is important to realize that having a solid testing plan means more than just hiring a dedicated tester, but also requires proper management and planning to insure that the testing is done the most effective way for each individual project.