UNIT – V

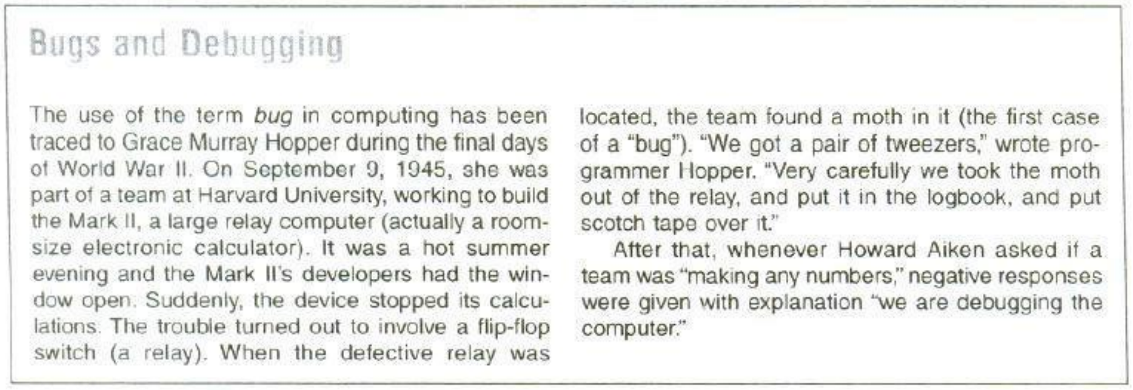
SOFTWARE QUALITY

Software quality assurance – Testing strategies – Test cases – Test plan – Myers debugging principle – System usability and measuring user satisfaction.

SOFTWARE QUALITY ASSURANCE

In the early history of computers, live bugs could be a problem (see Bugs and Debugging). Moths and other forms of natural insect life no longer trouble digital computers. However, bugs and the need to debug programs remain. In a 1966 article in *Scientific American,* computer scientist Christopher Strachey wrote,

Although programming techniques have improved immensely since the early years, the process of finding and correcting errors in prograrnming-"debugging" still remains a most difficult, confused and unsatisfactory operation. The chief impact of this state of affairs is psychological.



Bugs and Debugging

The use of the term *bug* in computing has been traced to Grace Murray Hopper during the final days of World War II. On September 9, 1945, she was part of a team at Harvard University, working to build the Mark II, a large relay computer (actually a room-size electronic calculator). It was a hot summer evening and the Mark II's developers had the window open. Suddenly, the device stopped its calculations. The trouble turned out to involve a flip-flop switch (a relay). When the defective relay was located, the team found a moth in it (the first case of a "bug"). "We got a pair of tweezers," wrote programmer Hopper. "Very carefully we took the moth out of the relay, and put it in the logbook, and put scotch tape over it."

After that, whenever Howard Aiken asked if a team was "making any numbers," negative responses were given with explanation "we are debugging the computer."

The elimination of the syntactical bug is the process of **debugging**, whereas the detection and elimination of the logical bug is the process of **testing**. Gruenberger writes, The logical bugs can be extremely subtle and may need a great deal of effort to eliminate them. It is commonly accepted that all large software systems(operating or application) have bugs remaining in them. The number of possible paths through a large computer program is enormous, and it is physically impossible to explore all of them. The single path containing a bug may not be followed in actual production runs for a long time (if ever) after the program has been certified as correct by its author or others.

QUALITY ASSURANCE TESTS

One reason why quality assurance is needed is because computers are infamous for doing what you tell them to do, not necessarily what you want them to do. To close this gap, the code must be free of errors or bugs that cause unexpected results, a process called debugging..

***Scenario-based testing**,* also called ***usage-based testing**,* concentrates on what the user does, not what the product does. This means capturing use cases and the tasks users perform, then performing them and their variants as tests. These scenarios also can identify **interaction bugs**. They often are more complex and realistic than **error-based tests**. Scenario-based tests tend to exercise multiple subsystems in a single test, because that is what users do. The tests will not find everything, but they will cover at least the higher visibility system interaction bugs .

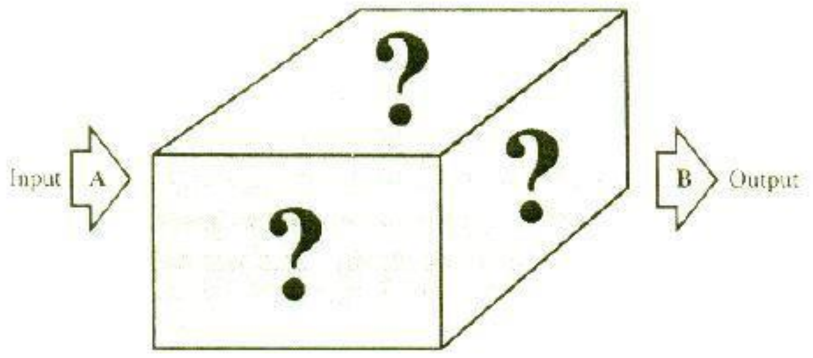
TESTING STRATEGIES

The extent of testing a system is controlled by many factors, such as the risks involved, limitations on resources, and deadlines. In light of these issues, we must deploy a testing strategy that does the "best" job of finding defects in a product within the given constraints. There are many testing strategies, but most testing uses a combination of these: black box testing, white box testing, top-down testing, and bottom-up testing. However, no strategy or combination of strategies truly can prove the correctness of a system; it can establish only its "acceptability."

Black Box Testing

The concept of the black box is used to represent a system whose inside workings are not available for inspection . In a black box, the test item is treated as "black," since its logic is unknown; all that is known is what goes in and what comes out, or the input and output (see Figure 13-1). Weinberg describes writing a user manual as an example of a black box approach to requirements. The user manual does not show the internal logic, because the users of the system do not care about what is inside the system.
In *black box testing,* you try various inputs and examine the resulting output; you can learn what the box does but nothing about how this conversion is implemented . Black box testing works very nicely in testing objects in an object-oriented environment. The black box testing technique also can be used for scenario-based tests, where the system's inside may not be available for inspection but the input and output are defined through use cases or other analysis information.

The black box is an imaginary box that hides its internal workings.



White Box Testing

White box testing assumes that the specific logic is important and must be tested to guarantee the system's proper functioning. The main use of the white box is in error-based testing, when you already have tested all objects of an application and all external or *public methods* of an object that you believe to be of greater importance (see Figure).

In white box testing, you are looking for bugs that have a low probability of execution, have been carelessly implemented, or were overlooked previously .

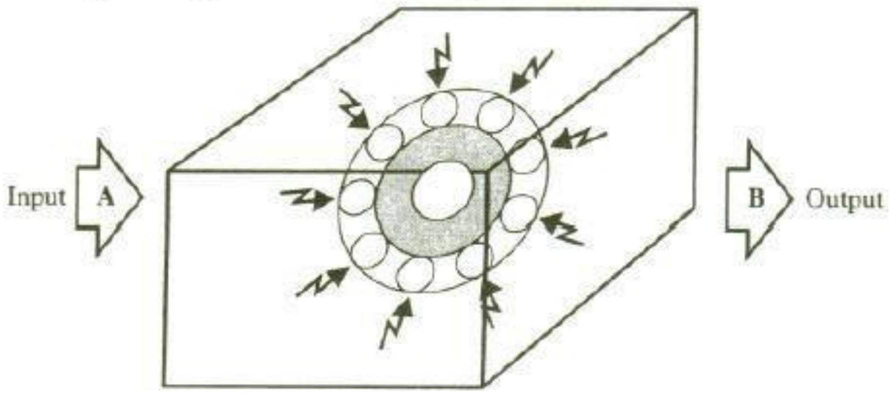
One form of white box testing, called ***path testing,*** makes certain that each path in a object's method is executed at least once during testing. **Two types of path testing** are statement testing coverage and branch testing coverage : .*Statement testing coverage.* The main idea of statement testing coverage is to test every statement in the object's method by executing it at least once. Murray states, "Testing less than this for new software is unconscionable and should be criminalized" [quoted in 2]. However, realistically, it is impossible to test a program on every single input, so you never can be sure that a program will not fail on some input. .*Branch testing coverage.* The main idea behind branch testing coverage is to perform enough tests to ensure that every branch alternative has been executed at least once under some test . As in statement testing coverage, it is unfeasible to fully test any program of considerable size.

Most debugging tools are excellent in statement and branch testing coverage. White box testing is useful for error-based testing.

Top-Down Testing

Top-down testing assumes that the main logic or object interactions and systems messages of the application need more testing than an individual object's methods or supporting logic. A top-down strategy can detect the serious design flaws early in the implementation.

In a white-box testing strategy, the internal workings are known.



In theory, top-down testing should find critical design errors early in the testing process and significantly improve the quality of the delivered software because of the iterative nature of the test . A top-down strategy supports testing the user interface and event-driven systems. Testing the user interface using a top-down approach means testing interface navigation. This serves two purposes, according to Conger. First, the top-down approach can test the navigation through screens and verify that it matches the requirements. Second, users can see, at an early stage, how the final application will look and feel . This approach also is useful for scenario-based testing. Topdown testing is useful to test subsystem and system integration.

Bottom-Up Testing

Bottom-up testing starts with the details of the system and proceeds to higher levels by a progressive aggregation of details until they collectively fit the requirements for the system. This approach is more appropriate for testing the individual objects in a system. Here, you test each object, then combine them and test their interaction and the messages passed among objects by utilizing the top-down approach.

In bottom-up testing, you start with the methods and classes that call or rely on no others. You test them thoroughly. Then you progress to the next level up: those methods and classes that use only the bottom level ones already tested. Next, you test combinations of the bottom two layers. Proceed until you are testing the entire program. This strategy makes sense because you are checking the behavior of a piece of codebefore it is used by another.Bottom-up testing leads to integration testing, which leads to systems testing.

TEST CASES

To have a comprehensive testing scheme, the test must cover all methods or a good majority of them.

All the services of your system must be checked by at least one test.

To test a system, you must construct some test input cases, then describe how the output will look.

Next, perform the tests and compare the outcome with the expected output.

The good news is that the use cases developed during analysis can be used to describe the usage test cases.

After all, tests always should be designed from specifications and not by looking at the product!

Myers describes the objective of testing as follows.

Testing is the process of executing a program with the intent of finding errors. A good test case is the one that has a high probability of detecting an as-yet undiscovered error. A successful test case is the one that detects an as-yet undiscovered error.

Guidelines for Developing Quality Assurance Test Cases

Gause and Weinberg provide a wonderful example to highlight the essence of a test case. Say, we want to test our new and improved "Superchalk": Writing a geometry lesson on a blackboard is clearly normal use for Superchalk. Drawing on clothing is not normal, but is quite reasonable to expect. Eating Superchalk may be unreasonable, but the design will have to deal with this issue in some way, in order to prevent lawsuits. No single failure of requirements work leads to more lawsuits than the confident declaration.

Basically, a test case is a set of what-if questions. Freedman and Thomas have developed guidelines that have been adapted for the UA: .Describe which feature or service (external or internal) your test attempts to cover. .If the test case is based on a use case (i.e., this is a usage test), it is a good idea to refer to the use-case name. Remember that the use cases are the source of test cases. In theory, the software is supposed to match the use cases, not the reverse. As soon as you have enough of use cases, go ahead and write the test plan for that piece. . Specify what you are testing and which particular feature (methods). Then, specify what you are going to do to test the feature and what you expect to happen. .Test the normal use of the object's methods. .Test the abnormal but reasonable use of the object's methods. .Test the abnormal and unreasonable use of the object's methods.

Test the boundary conditions. For example, if an edit control accepts 32 characters, try 33, then try 40. Also specify when you expect error dialog boxes, when you expect some default event, and when functionality still is being defined. .Test objects' interactions and the messages sent among them. If you have developed sequence diagrams, they can assist you in this process. .When the revisions have been made, document the cases so they become the starting basis for the follow-up test. .

.The internal quality of the software, such as its reusability and extendability, should be assessed as well. Although the reusability and extendability are more difficult to test, nevertheless they are extremely important. Software reusability rarely is practiced effectively. The organizations that will survive in the 21st century will be those that have achieved high levels of reusability-anywhere from 70-80 percent or more . Griss argues that, although reuse is widely desired and often the benefit of utilizing object technology, many object-oriented reuse efforts fail because of too narrow a focus on technology rather than the policies set forth by an organization. He recommends an institutionalized approach to software development, in which software assets intentionally are created or acquired to be reusable. These assets then are consistently used and maintained to obtain high levels of reuse, thereby optimizing the organization's ability to produce high-quality software products rapidly and effectively. Your test case may measure what percentage of the system has been reused, say, measured in terms of reused lines of code as opposed to new lines of code written. Specifying results is crucial in developing test cases. You should test cases that are supposed to fail. During such tests, it is a good idea to alert the person running them that failure is expected. Say, we are testing a File Open feature. We need to specify the result as follows:

1. Drop down the File menu and select Open.
2. Try opening the following types of files:

. A file that is there (should work).
.A file that is not there (should get an Error message).
.A file name with international characters (should work).
.A file type that the program does not open (should get a message or conversion dialog box).

TEST PLANS

On paper, it may seem that everything will fall into place with no preparation and a bug-free product will be shipped. However, in the real world, it may be a good idea to use a test plan to find bugs and remove them. A dreaded and frequently overlooked activity in software development is writing the test plan. A test plan offers a road map for testing activities, whether usability, user satisfaction, or quality assurance tests. It should state the test objectives and how to meet them. The test plan need not be very large; in fact, devoting too much time to the plan can be counterproductive.

The following steps are needed to create a test plan:

1. ***Objectives of the test***. Create the objectives and describe how to achieve them.
For example, if the objective is usability of the system, that must be stated and also how to realize it.
2.***Development of a test case.*** Develop test data, both input and expected output, based on the domain of the data and the expected behaviors that must be tested (more on this in the next section).
3. ***Test analysis***. This step involves the examination of the test output and the documentation of the test results. If bugs are detected, then this is reported and the activity centers on debugging. After debugging, steps 1 through 3 must be repeated until no bugs can be detected.

All passed tests should be repeated with the revised program, called *regression testing,* which can discover errors introduced during the debugging process. When sufficient testing is believed to have been conducted, this fact should be reported, and testing for this specific product is complete .

According to Tamara Thomas , the test planner at Microsoft, a good test plan is one of the strongest tools you might have. It gives you the chance to be clear with other groups or departments about what will be tested, how it will be tested, and the intended schedule. Thomas explains that, with a good, clear test plan, you can assign testing features to other people in an efficient manner. You then can use the plan to track what has been tested, who did the testing, and how the testing was done. You also can use your plan as a checklist, to make sure that you do not forget to test any features.

Who should do the testing? For a small application, the designer or the design team usually will develop the test plan and test cases and, in some situations, actually will perform the tests. However, many organizations have a separate team, such as a quality assurance group, that works closely with the design team and is responsible for these activities (such as developing the test plans and actually performing the tests). Most software companies also use *beta testing,* a popular, inexpensive, and effective way to test software on a select group of the actual users of the system. This is in contrast to *alpha testing,* where testing is done by inhouse testers, such as programmers, software engineers, and internal users. If you are going to perform beta testing, make sure to include it in your plan, since it needs to be communicated to your users well in advance of the availability of your application in a beta version.

GUIDELINES FOR DEVELOPING TEST PLANS

As software gains complexity and interaction among programs is more tightly coupled, planning becomes essential. A good test plan not only prevents overlooking a feature (or features), it also helps divide the work load among other people, explains Thomas.

The following guidelines have been developed by Thomas for writing test plans :
.You may have requirements that dictate a specific appearance or format for your test plan. These requirements may be generated by the users. Whatever the appearance of your test plan, try to include as much detail as possible about the tests. .The test plan should contain a schedule and a list of required resources. List how many people will be needed, when the testing will be done, and what equipment will be required.

After you have detennined what types of testing are necessary (such as black box, white box, top-down, or bottom-up testing), you need to document specifically what you are going to do. Document every type of test you plan to complete.

The level of detail in your plan may be driven by several factors, such as the following: How much test time do you have?

Will you use the test plan as a training tool for newer team members? .

A configuration control system provides a way of tracking the changes to the code. At a minimum, every time the code changes, a record shouh l, De kept that tracks which module has been changed, who changed it, and when it was altered, with a comment about why the change was made. Without configuration control, you may have difficulty keeping the testing in line with the changes, since frequent changes may occur without being communicated to the testers. .

A well-thought-out design tends to produce better code and result in more ,complete testing, so it is a good idea to try to keep the plan up to date. Generally, the older a plan gets, the less useful it becomes. If a test plan is so old that it has become part of the fossil record, it is not terribly useful. As you approach the end of a project, you will have less time to create plans . If you do not take the time to document the work that needs to be done, you risk forgetting something in the mad dash to the finish line. Try to develop a habit of routinely bringing the test plan in sync with the product or product specification. . At the end of each month or as you reach each milestone, take time to complete routine updates. This will help you avoid being overwhelmed by being so outof- date that you need to rewrite the whole plan. Keep configuration infonnation on your plan, too. Notes about who made which updates and when can be very helpful down the road

MYERS'S DEBUGGING PRINCIPLES

The Myers's bug location and debugging principles:
1. Bug Locating Principles . Think. . If you reach an impasse, sleep on it. . If the impasse remains, describe the problem to someone else. .Use debugging tools (this is slightly different from Myers's suggestion). .Experimentation should be done as a last resort (this is slightly different from Myers's suggestion).
2. Debugging Principles . Where there is one bug, there is likely to be another. .Fix the error, not just the symptom of it. .The probability of the solution being correct drops as the size of the program increases. .Beware of the possibility that an error correction will create a new error (this is less of a problem in an object-oriented environment).

CASE STUDY: DEVELOPING TEST CASES FOR THE VIANET BANK ATM SYSTEM

We identified the scenarios or use cases for the ViaNet bank ATM system. The ViaNet bank ATM system has scenarios involving Checking Account, Savings Account, and general Bank

Transaction (see Figures. Here again is a list of the use cases that drive many object-oriented activities, including the usability testing: .Bank Transaction (see Figure). .Checking Transaction History (see Figure). .Deposit Checking (see Figure).
.Deposit Savings (see Figure). .Savings Transaction History (see Figure). .Withdraw Checking (see Figure). .Withdraw Savings (see Figure). .Valid/Invalid PIN (see Figure).

The activity diagrams and sequence/collaboration diagrams created for these use cases are used to develop the usability test cases. For example, you can draw activity and sequence diagrams to model each scenario that exists when a bank client withdraws, deposits, or needs information on an account. Walking through the steps can assist you in developing a usage test case.

Let us develop a test case for the activities involved in the ATM transaction based on the use cases identified so far. (See the activity diagram in Figure and the sequence diagram of Figure to refresh your memory.)

SYSTEM USABLILITY AND USER SATISFACTION

INTRODUCTION

Quality refers to the ability of products to meet the users' needs and expectations. The task of satisfying user requirements is the basic motivation for quality. Quality also means striving to do the things right the first time, while always looking to improve how things are being done. Sometimes, this even means spending more time in the initial phases of a project-such as analysis and design-making sure that you are doing the right things. Having to correct fewer problems means significantly less wasted time and capital. When all the losses caused by poor quality are considered, high quality usually costs less than poor quality.

Two main issues in software quality are *validation* or user satisfaction and *verification* or quality assurance (see Previous chapter). There are different reasons for testing. You can use testing to look for potential problems in a proposed design. You can focus on comparing two or more designs to determine which is better, given
a specific task or set of tasks. Usability testing is different from quality assurance testing in that, rather than finding programming defects, you assess how well the interface or the software fits the use cases, which are the reflections of users' needs and expectations. To ensure user satisfaction, we must measure it throughout the system development with user satisfaction tests. Furthermore, these tests can be used as a communication vehicle between designers and end users . In the next section, we look at user satisfaction tests
that can be invaluable in developing high- Once the design is complete, you can walk users through the steps of the scenarios to determine if the design enables the scenarios to occur as planned.

USABILITY TESTING

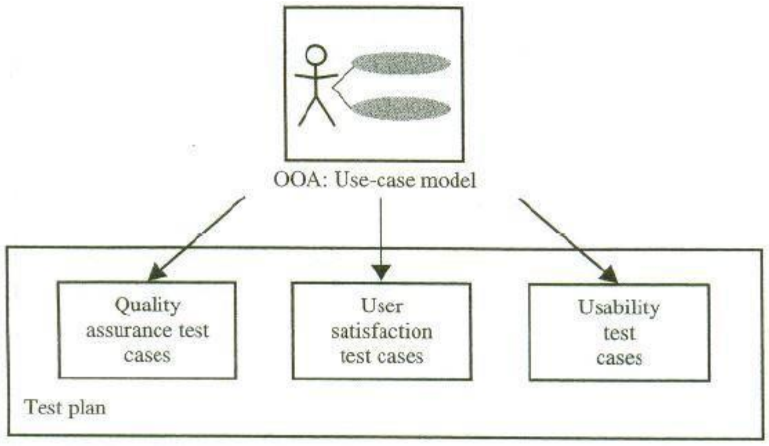
The International Organization for Standardization (ISO) defines *usability* as the effectiveness, efficiency; and satisfaction with which a specified set others can achieve a specified set of tasks in particular environments. The ISO definition requires .Defining tasks. What are the tasks? . Defining users. Who are the users? .*A means for measuring effectiveness, efficiency, and satisfaction.* How do we measure usability?

The phrase *two sides of the same coin* is helpful for describing the relationship between the usability and functionality of a system. Both are essential for the development of high-quality software . *Usability testing* measures the ease of use as well as the degree of comfort and satisfaction users have with the software. Products with poor usability can be difficult to learn, complicated to operate, and misused or not used at all. Therefore, low product usability leads to high costs for users and a bad reputation for the developers. Usability is one of the most crucial factors in the design and development of a product, especially the user interface. Therefore, usability testing must be a key part of the UI design process.

Usability testing should begin in the early stages of product development; for example, it can be used to gather information about how users do their work and find out their tasks, which can complement use cases. You can incorporate your findings into the usability test plan and test cases. As the design progresses, usability testing continues to provide valuable input for analyzing initial design concepts and, in the later stages of product development, can be used to test specific product tasks, especially the ill.

Usability test cases begin with the identification of use cases that can specify the target audience, tasks, and test goals. When designing a test, focus on use cases or tasks, not features. Even if your goal is testing specific features, remember that your users will use them within the context of particular tasks. It also is a good idea to run a pilot test to work the bugs out of the tasks to be tested and make certain the task scenarios, prototype, and test equipment work smoothly. Test cases must include all use cases identified so far. Recall from Previous chapter that the use case can be used through most activities of software development.

Furthermore, by following Jacobson's life cycle model, you can produce designs that are traceable across requirements, analysis, design, implementation, and testing. The main advantage is that all design traces directly back to the user requirements. Use cases and usage scenarios can become test scenarios; and therefore, the use case will drive the usability, user satisfaction, and quality assurance test cases (see Figure).



The use cases identified during analysis can be used in testing the design. Once the design is complete, walk users through the steps of the scenarios to determine if the design enables the scenarios to occur as planned.

GUIDELINES FOR DEVELOPING USABILITY TESTING

Many techniques can be used to gather usability information. In addition to use cases, focus groups can be helpful for generating initial ideas or trying out new ideas. A focus group requires a moderator who directs the discussion about aspects of a task or design but allows participants to freely express their opinions.

Usability tests can be conducted in a one-on-one fashion, as a demonstration, or as a "walk through," in which you take the users through a set of sample scenarios and ask about their impressions along the way. In a technique called the *Wizard of OZ,* a testing specialist simulates the interaction of an interface. Although these latter techniques can be valuable, they often require a trained, experienced test coordinator 9. Let us take a look at some guidelines for developing usability testing: .The usability testing should include aU of a software's components. . Usability testing need not be very expensive or elaborate, such as including trained specialists working in a soundproof lab with one-way mirrors and sophisticated recording equipment. Even the small investment of tape recorder, stopwatch, and notepad in an office or conference room can produce excellent results. . Similarly, all tests need not involve many subjects. More typically, quick, iterative tests with a small, well-targeted sample of 6 to 10 participants can identify 80-90 percent of most design problems. .Consider the user's experience as part of your software usability. You can study 80-90 percent of most design problems with as few as three or four users if you target only a single skill level of users, such as novices or intermediate level users. .

RECORDING THE USABILITY TEST

When conducting the usability test, provide an environment comparable to the target setting; usually a quiet location, free from distractions is best. Make participants

feel comfortable. It often helps to emphasize that you are testing the software, not the participants. If the participants become confused or frustrated, it is no reflection on them. Unless you have participated yourself, you may be surprised by the pressure many test participants feel. You can alleviate some of the pressure by explaining the testing process and equipment. . Tandy Trower, director of the Advanced User Interface group at Microsoft, explains that the users must have reasonable time to try to work through any difficult situation they encounter. Although it generally is best not to interrupt participants during a test, they may get stuck or end up in situations that require intervention. This need not disqualify the test data, as long as the test coordinator carefully guides or hints around a problem. Begin with general hints before moving to specific advice. For more difficult situations, you may need to stop the test and make adjustments. Keep in mind that less intervention usually yields better results. Always record the techniques and search patterns users employ when attempting to work through a difficulty and the number and type of hints you have to provide them.

Ask subjects to think aloud as they work, so you can hear what assumptions and inferences they are making. As the participants work, record the time they take to perform a task as well as any problems they encounter. You may want to follow up the session with the *user satisfaction* test (more on this in the next section) and a questionnaire that asks the participants to evaluate the product or tasks they performed.
Record the test results using a portable tape recorder or, better, a video camera.Since even the best observer can miss details, reviewing the data later will prove invaluable. Recorded data also allows more direct comparison among multiple participants. It usually is risky to base conclusions on observing a single subject. Recorded data allows the design team to review and evaluate the results.

Whenever possible, involve all members of the design team in observing the test and reviewing the results. This ensures a common reference point and better design solutions as team members apply their own insights to what they observe. If direct observation is not possible, make the recorded results available to the entire team. To ensure user satisfaction and therefore high-quality software, measure user satisfaction along the way as the design takes form . In the next section, we look at the user satisfaction test, which can be an invaluable tool in developing highquality software.

USER SATISFACTION TEST

INTRODUCTION

A positive side effect of testing with a prototype is that you can observe how people actually use the software. In addition to prototyping and usability testing, another tool that can assist us in developing high-quality software is measuring and monitoring user satisfaction during software development, especially during the design and development of the user interface.

USER SATISFACTION TEST

User satisfaction testing is the process of quantifying the usability test with some measurable attributes of the test, such as functionality, cost, or ease of use. Usability can be assessed by defining measurable goals, such as .95 percent of users should be able to find how to withdraw money from the ATM machine without error and with no formal training. .70 percent of all users should experience the new function as "a clear improvement over the previous one." . 90 percent of consumers should be able to operate the VCR within 30 minutes. Furthermore, if the product is being built incrementally, the best measure of user satisfaction is the product itself, since you can observe how users are using it-or avoiding it . Gause and Weinberg have developed a user satisfaction test that can be used along with usability testing. Here are the principal objectives of the user satisfaction test : .

As a communication vehicle between designers, as well as between users and designers. .To detect and evaluate changes during the design process. .To provide a periodic indication of divergence of opinion about the current design. .To enable pinpointing specific areas of dissatisfaction for remedy. .To provide a clear understanding of just how the completed design is to be evaluated.

Even if the results are never summarized and no one fills out a questionnaire, the process of creating the test itself will provide useful information. Additionally, the test is inexpensive, easy to use, and it is educational to both those who administer it and those who take it.

GUIDELINES FOR DEVELOPING A USER SATISFACTION TEST

The format of every user satisfaction test is basically the same, but its content is different for each project. Once again, the use cases can provide you with an excellent source of information throughout this process. Furthermore, you must work with the users or clients to find out what attributes should be included in the test. Ask the users to select a limited number (5 to 10) of attributes by which the final product can be evaluated. For example, the user might select the following attributes for a customer tracking system: ease of use, functionality, cost, intuitiveness of user interface, and reliability.

A test based on these attributes is shown in Figure . Once these attributes have been identified, they can playa crucial role in the evaluation of the final product. Keep these attributes in the foreground, rather than make assumptions about how the design will be evaluated . The user must use his or her judgment to answer each question by selecting a number between 1 and 10, with 10 as the most favorable and 1 as the least. Comments often are the most significant part of the test. Gause and Weinberg raise the following important point in conducting a user satisfaction test : "When the design of the test has been drafted, show it to the clients and ask, 'If you fill this out monthly (or at whatever interval), will it enable you to express what you like and don't like?' If they answer negatively then find out what attributes would enable them to express themselves and revise the test."

A TOOL FOR ANALYZING USER SATISFACTION: THE USER SATISFACTION TEST TEMPLATE

Commercial off-the-shelf (COTS) *software* tools are already written and a few are available for analyzing and conducting user satisfaction tests. However, here, I have selected an electronic spreadsheet to demonstrate how it can be used to record and analyze the user satisfaction test. The user satisfaction test spreadsheet (USTS) automates many bookkeeping tasks and can assist in analyzing the user satisfaction
test results. Furthermore, it offers a quick start for creating a user satisfaction test for a particular project.

Recall from the previous section that the tests need not involve many subjects. More typically, quick, iterative tests with a small, well-targeted sample of 6 to 10 participants can identify 80-90 percent of most design problems. The spreadsheet should be designed to record responses from up to 10 users. However, if there are inputs from more than 10 users, it must allow for that (see Figures).

One use of a tool like this is that it shows patterns in user satisfaction level. For example, a shift in the user satisfaction rating indicates that something is happening (see Figure . Gause and Weinberg explain that this shift is sufficient cause to follow up with an interview. The user satisfaction test can be a tool for

Measuring User Satisfaction
Project Name: Customer Tracking System

User	1	2	3	4	5	6	7	8	9
Ease of use	4	7							
Functionality	5	4							
Cost	1	6							
Realiablity	3	4							

Fig: User Satisfaction test for a Customer Tracing System

Periodical plotting can reveal shifts in user satisfaction, which can pinpoint a problem-Plotting the high and low responses indicates where to go for maximum information (Gause and Weinberg)

finding out what attributes are important or unimportant. An interesting side effect of developing a user satisfaction test is that you benefit from it even if the test is never administered to anyone; it still provides useful information. However, performing the test regularly helps to keep the user involved in the system. It also helps you focus on user wishes. Here is the user satisfaction cycle that has been suggested by Gause and Weinberg:

1. Create a user satisfaction test for your own project. Create a custom form that fits the project's needs and the culture of your organization. Use cases are a great source of information; however, make sure to involve the user in creation of the test.
2. Conduct the test regularly and frequently.
3. Read the comments very carefully, especially if they express a strong feeling.

Never forget that feelings are facts, the most important facts you have about the users of the system.

4. Use the information from user satisfaction test, usability test, reactions to prototypes, interviews recorded, and other comments to improve the product.

Another benefit of the user satisfaction test is that you can continue using it even after the product is delivered. The results then become a measure of how well users are learning to use the product and how well it is being maintained. They also provide a starting point for initiating follow-up projects.

CASE STUDY: DEVELOPING USABILITY TEST PLANS AND TEST CASES FOR THE VIANET BANK ATM SYSTEM

In previous previous chapter, we learned that test plans need not be very large; in fact, devoting too much time to the plans can be counterproductive. Having this in mind let us develop a usability test plan for the ViaNet ATM kiosk by going through the followings steps.

DEVELOP TEST OBJECTIVES

The first step is to develop objectives for the test plan. Generally, test objectives are based on the requirements, use cases, or current or desired system usage. In this case, ease of use is the most important requirement, since the ViaNet bank customers should be able to perform their tasks with basically no training and are not expected to read a user manual before withdrawing money from their checking accounts.

Here are the objectives to test the usability of the ViaNet bank ATM kiosk and its user interface:
95 percent of users should be able to find out how to withdraw money from the
ATM machine without error or any formal training. .90 percent of consumers should be able to operate the ATM within 90 seconds.

DEVELOP TEST CASES

Test cases for usability testing are slightly different from test cases for quality assurance. Basically, here, we are not testing the input and expected output but how users interact with the system. Once again, the use cases created during analysis can be used to develop scenarios for the usability test. The usability test scenarios are based on the following use cases:
Deposit Checking (see Figures).
Withdraw Checking (see Figures).
Deposit Savings (see Figures).
Withdraw Savings (see Figures).
Savings Transaction History (see Figures).
Checking Transaction History(see Figures).

Next we need to select a small number of test participants (6 to 10) who have never before used the kiosk and ask them to perform the following scenarios based on the use case:

1. Deposit $1056.65 to your checking account.
2. Withdraw $40 from your checking account.
3. Deposit $200 to your savings account.
4. Withdraw $55 from savings account.
5. Get your savings account transaction history.
6. Get your checking account transaction history.

Start by explaining the testing process and equipment to the participants to ease the pressure. Remember to make participants feel comfortable by emphasizing that you are testing the software, not them. If they become confused or frustrated, it is no reflection on them but the poor usability of the system. Make sure to ask them to think aloud as they work, so you can hear what assumptions and inferences they are making. After all, if they cannot perform these tasks with ease, then the system is not useful.

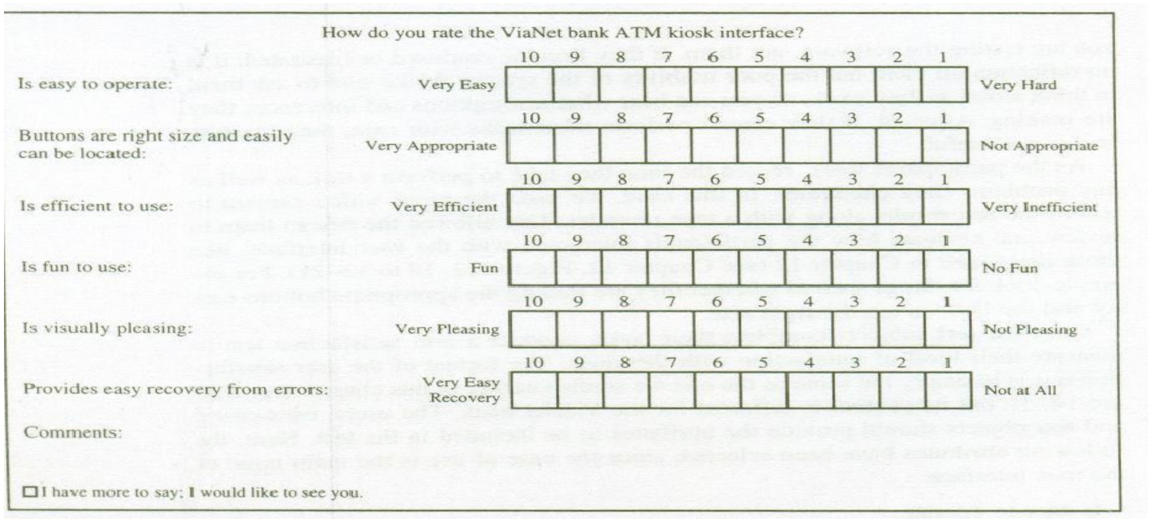
As the participants work, record the time they take to perform a task as well as any problems they encounter. In this case, we used the kiosk video...camera to record the test results along with a tape recorder. This allowed the design team to review and evaluate how the participants interacted with the user interface, like those developed in Previous chapter . For example, look for things such as whether they are finding the appropriate buttons easily and the buttons are the right size. Once the test subjects complete their tasks, conduct a user satisfaction test to measure their level of satisfaction with the kiosk.

ANALYZE THE TESTS

The final step is to analyze the tests and document the test results. Here, we need to answer questions such as these: What percentage were able to operate the ATM within 90 seconds or without error? Were the participants able to find out how to withdraw money from the ATM machine with no help? The results of the analysis must be examined.

We also need to analyze the results of user satisfaction tests. The USTS described earlier or a tool similar to it can be used to record and graph the results of user satisfaction tests. As we learned earlier, a shift in user satisfaction pattern indicates that something is happening and a follow-up interview is needed to find out the reasons for the changes. The user satisfaction test can be used as a tool for finding out what attributes are important or unimportant. For example, based 011 the user satisfaction test, we might find that the users do not agree that the system "is efficient to use," and it got a low score.

After the follow-up interviews, it became apparent that participants wanted, in addition to entering the amount for withdrawal, to be able to select from a list with predefined values (say, $20, $40).



How do you rate the ViaNet bank ATM kiosk interface?

		10	9	8	7	6	5	4	3	2	1	
Is easy to operate:	Very Easy											Very Hard
Buttons are right size and easily can be located:	Very Appropriate											Not Appropriate
Is efficient to use:	Very Efficient											Very Inefficient
Is fun to use:	Fun											No Fun
Is visually pleasing:	Very Pleasing											Not Pleasing
Provides easy recovery from errors:	Very Easy Recovery											Not at All

Comments:

☐ I have more to say; I would like to see you.