How to Write Better Test Cases

by Dianne L. Runnels, CQA, CSTE
Interim Technology Consulting

Investing in test cases

What is it worth to improve test cases? What risk would impel you to invest in better test cases? As long as they cover the software requirements, isn't that good enough? The answer to these questions is that poor test cases do indeed expose you to considerable risk. They may cover the requirements in theory, but are hard to test and have ambiguous results. Better tests have more reliable results as well as lowering costs in three categories:

1. Productivity - less time to write and maintain cases
2. Testability - less time to execute them
3. Scheduling reliability- better reliability in estimates

This paper describes how to avoid the losses that are inevitable with poor test cases. It will look under the hood of different kinds of test cases and show where and how to build in the quality that controls risk. It will give practical advice on how to improve productivity, usability, scheduling reliability, and asset management. Once you understand the whats and whys of test cases, you can use a checklist of standards, like the one attached as Appendix A, to identify areas of risk and improve your current and future test cases.

The most extensive effort in preparing to test software is writing test cases. The incentive to build robust test cases is spurred by the likelihood they will be reused for maintenance releases. Over half of all software development is maintenance projects. How can you write quality test cases that will deliver economical testing the first time plus live again as regression tests? Let's get started with the answer by lifting the hood of a test case and looking at what's inside.

Looking inside test cases

Elements of test cases

For our purposes, a test case is a set of actions with expected results based on requirements for the system. The case includes these elements:

- The purpose of the test or description of what requirement is being tested
- The method of how it will be tested
- The setup to test: version of application under test, hardware, software, operating system, data files, security access, time of day, logical or physical date, prerequisites such as other tests, and any another other setup information pertinent to the requirement(s) being tested
- Actions and expected results, or inputs and outputs
- Any proofs or attachments (optional)

These same elements need to be in test cases for every level of testing -- unit, integration, system, or acceptance testing. They are valid for functional, performance, and usability testing. The "expected results" standard does not apply to diagnostic or other testing of an exploratory nature. Even diagnostic testing needs the other elements in its cases. However, if the test measures performance that should fall in a range, this is an expected result.

An alternate description of test cases is that the description, purpose, and setup is the case or specification. The steps to accomplish it are called a script. Yet another view calls the purpose or description a scenario or use case. These views are all compatible with the quality assessments and improvements suggested in this paper.

Quality of test cases

There is a misconception that quality of writing is subjective, like looking at a painting, where beauty is in the eye of the beholder. In fact, quality of writing is objective and measurable. It is simple to set up an objective checklist, like the one in Appendix A, of the structural elements of test cases -- purpose,

method, setup, inputs and outputs. Then walk though each case. Is the element there or not? In addition to their structure, the cases must also meet these standards of quality:

Accurate. They test what their descriptions say they will test.

Economical. They have only the steps or fields needed for their purpose. They don't give a guided tour of the software.

Repeatable, self standing. A test case is a controlled experiment. It should get the same results every time no matter who tests it. If only the writer can test it and get the result, or If the test gets different results for different testers, it needs more work in the setup or actions.

Appropriate. A test case has to be appropriate for the testers and environment. If it is theoretically sound but requires skills that none of the testers have, it will sit on the shelf. Even if you know who is testing the first time, you need to consider down the road -- maintenance and regression.

Traceable. You have to know what requirement the case is testing. It may meet all the other standards, but if its result, pass or fail, doesn't matter, why bother?

Self cleaning. Picks up after itself. It returns the test environment to the pre-test state. For example, it doesn't leave the test system set at the wrong date. Automated scripts may call other scripts to do this. Don't confuse this standard with being destructive. Tests *should* be destructive, including trying to break a simulated production environment in controlled, repeatable ways.

These standards are also objective and measurable. They could also be added to your checklist. Someone who knows the requirements and application under test could fill out the checklist as part of a peer review. Compliance with some of the standards can't be known until after testing, but they are all measurable. This is an especially helpful exercise for new test writers to see where they consistently miss one of the elements, or don't meet a standard.

Format of test cases

What does a test case look like? They seem to fall into three major groups: step-by-step, matrix, and automated script. While the automated script will run as an online document, there is no assumption that the other two must be paper-based. They, too, might be online. Let's look at each of the formats:

Step-by-step. Figure one shows what the basics of this format looks like. A complete view of this format, in a template with other test elements is displayed as Appendix B.

Step	Action	Expected Result
1	Enter new name and address. Press <OK>.	Displays screen 008 new name details.
2	Fill all blanks with natural data. Make screen grab. Press <OK>.	Displays screen 005 maintenance.
3	Click on <Inquiry> button.	Displays screen 009 inquiry details.
4	Enter name from screen grab. Press <OK>.	Displays screen 010 record detail.
5	Compare record detail with screen grab.	All details match exactly.

Figure 1 - Detail of step-by-step test case

Matrix or table. Figure two shows what the basics of this format looks like. A complete view of the format, in a template with other test elements is displayed as Appendix C.

Date	Hired After 1/96	401K	Life Insurance	Payment Computation
10/25/99	Y	1	3	$24.50
1/4/98	Y	3	1	$34.00
3/6/96	N	2	5	$48.00
8/15/96	Y	2	5	$86.25
8/15/96	N	2	5	$105.00

Figure 2 - Detail of matrix test case

Automated script. Figure three shows what this format looks like.

```
# Open the Fax Form
        menu_select_item ("File;Fax Order...");
        set_window("Fax Order");

        # Retrieve the Fax Order information and compare it to data from the main window
        edit_get_text("Arrival:", text);
        if(main_data["arr_time"] != text)
        {
           failure_msg = arrival_fr_mismatch;
           result = FAIL;
```

Figure 3 - Detail of automated script test case

Using test types

Best uses for each type of case

The most productive uses for step-by-step cases are:

Step-by-step

- One-off test cases, each one different
- Business scenario goes from screen to screen
- Many processing rules
- GUI interfaces
- Input and output hard to represent in a matrix

Step-by-step cases do not necessarily need to have a key press level of detail as shown in Figure 1. The action steps can be at a higher level, such as: open "my account" page, search for transactions in a date range, note the range: _____ , print report, forward report, and so forth.

Matrix. The most productive uses for matrix cases are:

- Many variations of filling out a form, same fields, different values, input files
- Same inputs, different platforms, browsers, configurations
- Character based screens
- Input and outputs best represented in a matrix

Nearly any test can be represented in a matrix, but the question to decide is whether a matrix is the best way to test. It is most important that the matrix be supported by a description, setup, and how to record results for the test. It may be obvious to the writer how to test the matrix, but without more guidance the tester could flounder.

A variation of the matrix is a list of inputs. It can be inserted in a step-by-step test or stand as a matrix with the explanatory elements of the test.

Automated scripts. A decision to use automated testing software is more related to the project and organization doing the testing than to what is being tested. There are some technical issues that must be met, varying from tool to tool, but most applications can find a technical fit. The project management must understand that writing automated cases takes longer than manual tests because the manual tests must be still be written first. When the interface is stable, then the tests can be recorded. The real payback of automated testing comes in the maintenance phase of the software lifecycle. Then the scripts can be executed repeatedly, even unattended, for great savings in testing time.

The management must provide someone to program in the tool language, such as C++ or Visual Basic. Simple scripts can be recorded by most test analysts, but more powerful scripts, often utilizing custom

functions and objects, must be written by programmers. A team can be put together with several people doing the ordinary record/playback scripting and one person doing the programming.

Comparisons to the other types of test cases apply to record and playback, or data driven scripts. Besides record/playback scripts, automated tools are used for performance and load testing. They may use manual step-by-step cases or matrixes which detail how automated tools will be used to create virtual users, launch transaction scripts, monitor performance, and other activities.

Choosing a test type

The preference for one type of test case over another is driven as much by the culture and perceptions of the organization as by what is the best fit for the software and test plan. "We've always done it this way" is joined by some myths that die hard:

Myth one. Step-by-step test cases take too long to write. We can't afford them.
Reality: They may or may not take longer to write, and granularity makes them sturdy and easy to maintain. They are the only way to test some functions adequately.

Myth two. A matrix is always the best choice. Make it work.
Reality: A persistent problem is putting together a matrix with proper set-up information. Too often this information is omitted, or worse yet, if different setups or classes of input can't be forced into a matrix with a like group, they are not tested at all

Myth three. High tech is best. If you can automate test cases, do it.
Reality: A decision to use automated testing should be based on many factors.

Myth four. We don't have time to write manual test cases. Let's automate them.
Reality: Automated test cases take longer to create than the other two types.

An immature organization -- one that depends more on personalities than process -- is most apt to assume these myths are true if a dominant person mouths them. Their road to quality is going to be a long one. They may continue with misconceptions about test cases until they suffer a disappointing schedule or budget loss that is obviously due to poor tests.

Another inhibitor to using the best type of test case for the job is the tension between verbal and numeric expression. If you excel in one or the other, you probably prefer the type of test case that you can do intuitively. Step-by-step cases tend to be more verbal, and matrixes more numeric. Good training should build understanding and confidence to use all types of cases, each where it is most productive. However, personal preference is a factor to be reckoned with. If you are a manager, you need to win through by education and example. Often the most productive route is to use all three types of cases. The first two for unit, integration, and system testing; and automated scripts for regression testing.

Improving test cases

Improving testability of test cases

The definition of testability is easy to test -- accurately. Easy can be measured by how long it takes to execute the test, and whether the tester has to get clarification during the testing process. Accurately means that if the tester follows the directions, the result of pass or fail will be correct.

Improving testability with language

Test steps should be written in active case. Tell the tester what to do. Examples:

- Do this. Do that.
- Navigate to the shopping cart page.
- Compare the items in the cart with the screen capture.
- Click on <OK>.

It should always be clear whether the tester is doing something or the system is doing it. If a tester reads, "The button is pressed," does that mean he or she should press the button, or does it mean to verify that the system displays it as already pressed? One of the fastest ways to confuse a tester is to mix up

actions and results. In Figure 1, actions always go on the left side, results on the right side. What the tester does is always an action. What the system displays or does is always a result.

An easy habit to fall into if we know a subject inside and out is to refer to the same thing in different ways. The language of a test has to be scrupulous in naming screens, fields, servers, Web pages, or other test objects. In a development team we get into the habit of referring to certain objects generically before they are even in prototype, such as "the e-mail reply screen," when its label will actually say "order confirmation." Or we could refer to a screen by number, when the number doesn't appear anywhere the tester could see it. The test has to have exact references to orient the tester.

Testing can be accelerated if you build in structured fields for the tester to note inputs that will be verified later. For example, if you are testing an audit log, you can't know in advance the exact minute the tester will complete an auditable action. If you tell them to note it, it might be scribbled on the test, on a scrap of paper, or put in a temporary file. They probably won't take the trouble to mark what's what, and may have to search around to find it. Better to leave a labeled space in the test where they can write it down or type it in. Example:

Delete item date:_____
Time: _____
Item ID:_____
Tester login ID:_____

Improving testability by controlling length

How long should a test case be? A good length for step-by-step cases is 10-15 steps, unless the tester can't save their work. There are several benefits to keeping tests this short:

- It takes less time to test each step in short cases
- The tester is less likely to get lost, make mistakes, or need assistance.
- The test manager can accurately estimate how long it will take to test
- Results are easier to track

Don't try to cheat on the standard of 10-15 steps by cramming a lot of action into one step. A step should be one clear input or tester task. You can always tag a simple finisher onto the same step such as click <OK> or press <Enter>. Also a step can include a set of logically related inputs. You don't have to have a result for each step if the system doesn't respond to the step.

The goal of keeping individual tests short is not at odds with the traditional goal to test an application with the minimum amount of tests. The standard that the traditional goal aims at is to be clever and work into the 10-15 steps as many requirements as make sense for the business scenario or use case of the test. The standard also aims to avoid duplication, meaning testing the same requirement in a number of tests. You don't want to create a minimum number of tests by making each one so long that it is a nightmare to test or maintain. A better way to look at the old "minimum number of tests," gauge would be a "minimum number of steps." Then there is no perceived advantage to creating fewer tests by putting 50 steps in each.

A good length for a matrix is what can be tested in about 20 minutes.

The length of an automated script is not a test execution issue, since testing is usually a matter of seconds. Rather, the issues are management of content, maintenance, and defect tracking. One business scenario or use case per script is enough. It can loop as many times as necessary to load data or exercise variable combinations.

Pros and cons of cumulative cases

Cumulative cases are those that depend on previous cases to run -- when you have to run tests 1-5 before you can test #6. Your goal is to keep the test as self-standing as possible. This gives the greatest flexibility in scheduling testing, and cuts down on maintenance time. Unfortunately, the down side to this is that the standard may be at odds with other standards, such as keeping each test case short and not duplicating coverage. How do you achieve this?

- Ask yourself if you really need data input from another test? If so, the test must be cumulative.
- Whenever possible, offer an alternative to a previous test. This implies they can use the data created in a previous test but they could also use other data. Example: "You need two accounts in 90-day delinquency status, such as the ones created for the overdue accounts test cases."
- Keep references to other tests as generic as possible consistent with accuracy. Don't refer to tests by number only. Tests get renumbered. If you use a number, include also the title or description. This can avoid a maintenance nightmare.

Another issue in the ordering of test cases to improve usability or testability is that they should be ordered by business use. What would the end user typically do first, not just what they have to do first, as in cumulative cases? If the tester is a business user, they will have a mental model of the tasks they want to accomplish with the software. It increases their testing productivity to orient them by reproducing their model.

Improving productivity with templates

A test case template is a form with labeled fields. There is a template for a step-by-step test case attached as Appendix B. This is a great way to start improving test cases. It jump starts the writing process and supports each of the elements of a good case. Here are some other benefits of using templates:

- Prevents blank page panic
- Assists the disorganized
- Builds in standards
- Prints spiffy looking tests
- Assists testers to find information
- Can include other fields relating to testing process

Improving productivity with clones

Cloning test cases means to model one test case on another one. A case is a good candidate for cloning if it fits the need for a step-by-step case but has variables that can easily be substituted. For example, you may have tests for maintaining a supplier database. Many, but not all, the steps would also apply to a shipper database. As you get to know the software through requirements or prototypes, strategize which functions work in such a way that you can clone the test cases. Writing them as clones does not mean they need to be tested together. You can clone steps as well as test cases.

Word processing and test authoring software support cloning with features such as "Save As," "Copy," and "Replace." It's very important to proofread these cases to make sure all references to the original are replaced in the clone. You can also save time with stored text and macros, but don't try to make one size fit all at the expense of precision. Look around your project for cloning opportunities such as other people's cases, user manual or help desk tutorials. They may be looking to trade with you for test cases. If you are managing a writing team, keep a test repository current among the writers so they can piggyback and share cases.

Matrixes can also be cloned, especially If the setup section is the same. The variables would be changes in the field names and values. Again, make sure to proofread the new version.

Improving productivity with test management software

Software designed to support test authoring is the single greatest productivity booster for writing test cases. It has these advantages over word processing, database, or spreadsheet software:

- Makes writing and outlining easier
- Facilitates cloning of cases and steps
- Easy to add, move, delete cases and steps
- Automatically numbers and renumbers
- Prints tests in easy-to-follow templates

Test authoring is usually included in off-the-shelf test management software, or it could be custom-written. Test management software usually contains more features than just test authoring. When you

factor them into the purchase, they offer a lot of power for the price. If you are shopping for test management software, it should have all the usability advantages listed just above, plus additional functions:

- Exports tests to common formats
- Multi-user
- Tracks test writing progress, testing progress
- Tracks test results, or ports to database or defect tracker
- Links to requirements and/or creates coverage matrixes
- Builds test sets from cases
- Allows flexible security

Mistakes and challenges

The seven most common test case mistakes

In each writer's work, test case defects will cluster around certain writing mistakes. If you are writing cases or managing writers, don't wait until cases are all done before finding these clusters. Review the cases every day or two, looking for the faults that will make the cases harder to test and maintain. Chances are you will discover that the opportunities to improve are clustered in one of the seven most common test case mistakes:

1. Making cases too long
2. Incomplete, incorrect, or incoherent setup
3. Leaving out a step
4. Naming fields that changed or no longer exist
5. Unclear whether tester or system does action
6. Unclear what is a pass or fail result
7. Failure to clean up

Handling challenges to good test cases

Even when you employ the best techniques and standards, you still have to overcome the same challenges that face every test writing effort. Let's look at common challenges to test writing, and see how they can be managed by responses that salvage as much quality as possible. The challenges typically are imposed at the project level, and must be responded to at the testing management level. If they are imposed on a writer from the testing management level, the writer should make the response.

Challenge: Requirements changes
Response:

1. The best defense here is being informed. Before writing cases, and at every status meeting, find out where the greatest risk of requirement changes are. Strategize what cases will and won't be affected by the change. Write the ones that won't first.
2. Build in variables or "to be decided" that you will come back and fill in later.
3. Make sure the budget owner knows the cost of revising test cases that are already written. Quantify what it costs per case.
4. Let project management set priorities for which cases should be written or revised. Let them see you can't do it all and ask them to decide where they have greatest risk.
5. Release the not-quite-right test cases unrevised. Ask the testers to mark up what has to be changed. Schedule more time to test each case, plus time for maintaining the tests.

Challenge: Schedule changes
Response:

1. If a testing date is moved up, get management to participate in the options of how test cases will be affected. As in the changing requirements challenge, let them choose what they want to risk.
2. Add staff only if time permits one to two weeks of training before they have to be productive, and only if you have someone to mentor and review their work.
3. Shift the order of writing cases so you write those first that will be tested first. Try to stay one case ahead of the testers.

4. You can skinny down the test cases to just a purpose, what requirement is being tested, and a setup. This is not as bad as ad hoc testing, but management should know the results are not as reliable as if the cases were complete. Schedule more time to test this kind of test case, and time to finish the case after testing.
5. Offer to have writers do the testing and write as they go. Schedule more time for testing and finishing the writing after testing.

Challenge: Staff turnover
Response:
1. New staff need an understanding of the goals, schedule, and organization of the current testing project, in writing if possible. Verbal orientations get lost in the shuffle.
2. New staff should concentrate on knowing the business use of the software, and then on the requirements and prototypes. They may write fewer cases, but they will be right.
3. New staff should have hands-on training in the standards, with many practical examples of how to apply it. Their work should be closely checked at first.
4. Try to place new staff in an area of good technical fit for the cases they will be writing.

Test case assets

Protecting test case assets

The most important activity to protect the value of test cases is to maintain them so they are testable. They should be maintained after each testing cycle, since testers will find defects in the cases as well as in the software. When testing schedules are created, time should be allotted for the test analyst or writer to fix the cases while programmers fix bugs in the application. If they aren't fixed, the testers and writers will waste time in the next cycle figuring out whether the test case or the software has the error.

Test cases lost or corrupted by poor versioning and storage defeat the whole purpose of making them re-usable. Configuration management (CM) of cases should be handled by the organization or project, rather than the test management. If the organization does not have this level of process maturity, the test manager or test writer needs to supply it. Either the project or the test manager should protect valuable test case assets with the following configuration management standards:

- Naming and numbering conventions
- Formats, file types
- Versioning
- Test objects needed by the case, such as databases
- Read only storage
- Controlled access
- Off-site backup

Test management needs to have an index of all test cases. If one is not supplied by CM, create your own. A database should be searchable on keys of project, software, test name, number, and requirement. A full-text search capability would be even better.

Leveraging test cases

Test cases as development assets have a life beyond testing. They represent a complete picture of how the software works written in plain English. Even if the focus is destructive, they must also prove that all business scenarios work as required. Often the cases are written for testers who are the business users so they use real world language and terms. A set of use cases has tremendous value to others who are working to learn or sell the software:

- Business users
- Technical writers
- Help desk technicians
- Trainers
- Sales and marketing staff
- Web administrators

All of these people have a stake in seeing the software succeed, and are also potential testers. Depending on the organization, good will and open communication between test writers and these groups can greatly speed up the time to production or release.

Summary

The process of teaching good writing techniques and setting test case standards is an asset in itself. It is never static, but must be dynamically taught, applied, audited, measured, and improved. This paper has briefly covered what the processes and standards of test case quality are, how to apply them to all kinds of test cases, how to use them to improve testability and productivity, how to solve common challenges to test case quality, and how to protect test case assets.

Case study

This is a story of how five software quality analysts learned that good test cases make a measurable difference. They were an experienced, self-managed group, each with a different idea of what a test case should look like. A single case ranged from dozens of rambling, wordy directions, to a matrix with no directions at all. The testers were business users with little software confidence. They were eager to test, but after a week of spending more time trying to figure out what to do than actually testing, they pretty much threw in the towel. Their manager finally told the analysts they had put in the amount of time they agreed to, and had no more time to give. At this point less than half the tests were executed. Of these, many results were just question marks. One of the testers had even written angry remarks on the tests. The analysts figured the testers were just whiners and not very smart.

During this miserable cycle, the analysts got a new test manager. She looked at the test cases, and began a training and mentoring program. She demonstrated what good test cases looked like, gave them a checklist, and made the group use it to write cases for the next software module. It was scheduled for testing in two months. She met one on one with each of them and gave them specific help how to improve. Every one of the writers began to produce cases that met the standards. The first week of writing was slow going, but at the end of the two-month period they were all meeting a productivity goal. By the next testing cycle the cases were shorter, each with a clear purpose and precise pass or fail criteria. Still the analysts worried that testing was going to be troublesome again.

The testers started out expecting another negative experience, but quickly changed their attitude. They found they could easily complete each case without having to call or come over to the writers' cubicles. Their confidence grew, and some said they had been dreading the changeover to this software, but now they could see it was better than what they were using. The word got out to the sales people, who had been eager to learn about the software. Their manager came and asked if they could test also. The test manager didn’t need more testers for that cycle, but signed them up for the next one. The technical writers asked if they could have copies of the cases too.

The test manager kept some metrics on the improvements. When she first arrived, the analysts were spending 2 to 3 hours a day troubleshooting the bad cases with testers. They had scheduled the testing to take 20 minutes a test when in fact they were averaging about 45 minutes. In addition to the time spent with testers, some analysts were spending another hour or two daily trying to fix cases in the current cycle instead of their scheduled work -- writing cases for the next module.

After the training and standard setting, the metrics for the next testing cycle looked much better. None of the analysts spent more than one hour a day helping testers. Even though there were more tests, due to the standard of shorter test cases, testers could finish them in 20 minutes, and testing often was over a day early. By the end of the project, the analysts and testers were credited with speeding up the release by a month.

Appendix A

Test Case Checklist

Quality Attributes

- ☐ Accurate - tests what the description says it will test.
- ☐ Economical - has only the steps needed for its purpose
- ☐ Repeatable, self standing - same results no matter who tests it.
- ☐ Appropriate - for both immediate and future testers
- ☐ Traceable - to a requirement
- ☐ Self cleaning - returns the test environment to clean state

Structure and testability

- ☐ Has a name and number
- ☐ Has a stated purpose that includes what requirement is being tested
- ☐ Has a description of the method of testing
- ☐ Specifies setup information - environment, data, prerequisite tests, security access
- ☐ Has actions and expected results
- ☐ States if any proofs, such as reports or screen grabs, need to be saved
- ☐ Leaves the testing environment clean
- ☐ Uses active case language
- ☐ Does not exceed 15 steps
- ☐ Matrix does not take longer than 20 minutes to test
- ☐ Automated script is commented with purpose, inputs, expected results
- ☐ Setup offers alternative to prerequisite tests, if possible
- ☐ Is in correct business scenario order with other tests

Configuration management

- ☐ Employs naming and numbering conventions
- ☐ Saved in specified formats, file types
- ☐ Is versioned to match software under test
- ☐ Includes test objects needed by the case, such as databases
- ☐ Stored as read
- ☐ Stored with controlled access
- ☐ Stored where network backup operates
- ☐ Archived off-site

Appendix B

Test Case Template

Project No: **not assigned**	Project Name: **E-Commerce Banking Upgrades**		Page___1___of__1____
Case No./Name: **Transmit files**		Run No/Name: **1.1.15/Xnet receives comms**	Test Status:
Build/Module/Level/Function/Code Unit under Test: **Communications module/Integration testing/Xcom scripts**		Requirement No/Name: **From comms section, project SOW**	
Written by: **PW**	Date: **9-28-98**	Executed by: **unassigned**	Date:

Test Case Description (purpose and method): **Tests Remoteware sweep of incoming client transaction to mailbox on RS6000. You will initiate transmission, capture file names, check A/T/B log, and confirm with the technology center that the sweep got the application into the RS6000 mailbox.**

Setup for Test (hardware, software, data, prerequisite tests, timing, security): **Sweep runs continuously every 30 seconds from 8 a.m. to 5 p.m. Transmit one credit application from client to the bank. Any application from previous test cases will work. Contact the technology center to make sure you will have access to the test unit mailbox when you plan to run this test. Also, they will need to stand by to verify to you that the files have moved into the mailbox.**

Step	Action	Expected Result	Pass / Fail
1	**Open Remoteware screen before connecting to the bank.**	**System displays empty screen where you will be able to view the files as they come into the bank.**	
2	**Start E-Commerce screen, go to dial function, and connect to the bank.**	**System displays the number it is dialing.**	
3	**Switch to Remoteware screen.**	**Once connected, the system displays the .syn file to verify that E-Commerce is correct version. Then it sends the 'ill files.**	
4	**Write the names of the .ill files here: _____, _____, _____**	**After a successful transmission, the Remoteware node closes.**	
5	**Switch back to E-Commerce screen. Click the <Communications> button.**	**Displays Communications menu.**	
6	**Click the <A/T/B> button.**	**Displays A/T/B register.**	
7	**Find the transmission log for this test case. Print it using icon on the toolbar.**		
8.	**Verify the .ill files from this test case were sent.**	**A/T/B log shows transmission.**	
9.	**Call the technology center. Ask them to check their log to see if it matches the A/T/B log.**	**The entries should match for 1) time 2) length 3) file names**	
10	**Ask technology center to verify that message was swept to customer mailbox on RS/6000.**	**Client application is in their mailbox.**	

Matrix Template

Project Name: TST Enhancements	Project No: MIT3012	Page 1 of 1
Test Name: HR User rights matrix		
Build No: 4	Run No: 16 Case No:1	Execution date: By:
Written by: DLR	Date: 2/4/98	Requirement No: TST81
Test Case Description (purpose and method) : Tests security combinations in HR maintenance module. Tester creates users with rights as in matrix. Login as each user and attempt ALL actions, both granted and restricted. Mark matrix with results. Delete users that pass. Save users that fail. Move saved users into RBD database. Fill out RBD form with fail reasons for each user profile.		
Setup for Test: DBA must load HR maintenance database for version under test. Tester login ID must be "MASTER." Get daily password from test manager. Check for and delete any user profiles left in system. Access to test unit goes down at 5:30 p.m. No weekend access.		

Pass/ Fail	Test User #	View	Add	Edit	Appv	Delete	Archive	Audit	Reports	Security	Rebuild
	1	X									X
	2	X	X	X	X	X	X	X	X	X	X
	3	X	X	X		X	X	X	X		X
	4	X	X	X			X	X		X	X
	5	X							X		X
	6	X		X	X						X
	7	X		X					X		X
	8	X				X	X	X	X		X
	9	X			X						X
	10	X								X	X
	11	X		X		X					X
	12	X						X			X