STARWEST 2000

Creating a Test Plan Database for Standardized Tests Across Multiple Nodes: Removing yourself from the Database Role

Colleen L. Sherman
October 27, 2000

Return to Table of Contents

Table of Contents

Return to Table of Contents


I) Overview

This is a presentation of a database for test plans as not seen in some of the popular testing tools on the market today. The focus is geared towards Web testing of an application with nodes of similar properties; however, this concept can be applied to Client Server testing and other testing as well. If this concept exists in a tool somewhere out there already, then I've reinvented the wheel, but so far I have not seen it! Most test databases on the market that I've seen offer the ability to enter tests in a linear style, without the ability to repeat easily the same tests for many different nodes. If you have a large quantity of nodes that you need to test in a similar way, why not create a 2 dimensional storage facility with tests on one axis, and nodes on another, combined with different views for the testing lead, tester, program lead, and management.

This presentation will take the participant on a journey through the long-winded effort of a manual test-tracking approach that transformed the testing lead's job into an administrative nightmare. The original testing approach offered test security to the project, but took up unnecessary time that could have been better spent on actually testing the application! The concepts were all there from the start, and only through the manual testing coordination efforts was it discovered that there is a better way – the database way! See the old approach first, and the extensive administrative labor involved, then witness the combination of necessary testing procedures with the efficient and accurate solution that a test plan database can offer.

Return to Table of Contents


II) Sample Application

First let's examine a fictitious Web Financial Application to use as our model throughout the presentation…

A) Modules

This application has 4 Modules, or categories, and is accessible in the form of a TreeView, or accordion format. The Modules are "Payments", "Receipts", "Reports", and "Maintenance".

Financial Application
- Payments
- Receipts
- Reports
- Maintenance

B) Nodes

If you expand the TreeView, you can see that the "Payments" Module has 3 Nodes, or sub-categories. The Nodes are "Create Payment", "Cancel Payment", and "View Payment".

Financial Application
- Payments
  - *Create Payment*
  - *Cancel Payment*
  - *View Payment*
- Receipts
- Reports
- Maintenance

C) Actions

Expanding the "Create Payment" Node, you can see that there is one Action in this Node. The Action is "Confirm".

Financial Application
- Payments
  - *Create Payment*
    - *Confirm*
  - *Cancel Payment*
  - *View Payment*
- Receipts
- Reports
- Maintenance

Return to Table of Contents

D) Expanded TreeView

Likewise, if you expand the entire TreeView of the application, you can see all of the Modules, Nodes, and Actions.

Financial Application

- Payments
  - *Create Payment*
    - *Confirm*
  - *Cancel Payment*
  - *View Payment*
- Receipts
  - *Create Receipt*
    - *Confirm*
  - *Cancel Receipt*
  - *View Receipt*
- Reports
  - *Payment Report*
  - *Receipt Report*
  - *Summary Report*
- Maintenance
  - *Edit Accounts*
  - *Edit Currency*
  - *Add Users*
  - *Set System Date*
    - *Set*

E) Notes

Please keep in mind that this sample application has 4 Modules, 13 Nodes, and 3 Actions. This concept was designed for a project with 4 applications, the largest having approximately 10 Modules, 200 Nodes, and 50 Actions. So while a manual test tracking approach may seem manageable for a small application, it is not suitable for a large application. A database solution is needed!

Return to Table of Contents

III) Sample Tests

Let's look at some sample tests to be used as examples in this presentation, and how the tests apply to different types of Nodes.

A) Different Tests for Different Types of Nodes

In the sample Financial Application, there are different types of Nodes. Some Nodes are Interactive, meaning the user has the ability to add new records, edit existing records, or delete records (example "Create Payment"). Some Nodes are Inquiry, meaning the user can only view the records (example "View Payment").

Because Interactive Nodes have more functionality for the user, more tests need to be performed on such nodes. Subsequently, fewer tests need to be performed on Inquiry Nodes. For example, testing "Required Fields", which is trying to save a record with system-defined required fields null, needs to be performed on Interactive Nodes, but does not need to be performed on Inquiry Nodes because the Inquiry nodes do not offer the ability to save a record.

B) Test Descriptions

Below is a list of sample tests and their descriptions. Also listed are the types of nodes to which the tests are relevant.

Test	Description	Relevant to following Node types
1) Required Fields	Add or edit record. Try saving record with first required field null. Verify that message pops up "Required Field missing". Perform this test on all required fields in node.	Interactive
2) DropDowns	Examine DropDown fields in records. Verify that there are horizontal and vertical scroll bars, if necessary, to navigate through DropDown list. Verify that columns in DropDown list have headings.	Interactive
3) Tab Sequence	Tab through all fields of search criteria and records.	Interactive, Inquiry
4) LookUps	Verify that LookUp window pops up, and fields populate correctly in criteria and records.	Interactive, Inquiry
5) RadioButtons	Test functionality of Radio Buttons in records.	Interactive
6) Color	Examine color of entire node… window, fields, field labels, etc.	Interactive, Inquiry
7) Font	Examine font of node… field labels, record and criteria labels, etc.	Interactive, Inquiry
8) Add New Record	Add new record in node, and retrieve record. Verify record contents.	Interactive

Return to Table of Contents

IV) Original Test Plan Method

The original test plan method of the project for which this concept was designed is as follows.

A) Excel Spreadsheet

An Excel spreadsheet was created with the application's Modules, Nodes, and Actions listed vertically, and tests listed horizontally. Unfortunately, an 8 ½ by 11 inch, or even 14 inch document is only so wide, so only a certain number of the desired tests fit on the document. Consequently, a subsequent document was created listing the tests not shown here, with a column "More Tests" on the first document to reference the completion of the tests on the subsequent document. Cumbersome! A column was also added called "Test Complete", symbolizing that all tests have been completed on that node.

Financial Application	**TESTS**					
	Required Fields	Drop Downs	Tab Sequence	LookUps	More Tests	Test Complete
• Payments						
• *Create Payment*						
• *Confirm*						
• *Cancel Payment*						
• *View Payment*						
• Receipts						
• *Create Receipt*						
• *Confirm*						
• *Cancel Receipt*						
• *View Receipt*						
• Reports						
• *Payment Report*						
• *Receipt Report*						
• *Summary Report*						
• Maintenance						
• *Edit Accounts*						
• *Edit Currency*						
• *Add Users*						
• *Set System Date*						
• *Set*						

B) Doctoring the excel spreadsheet

Because some of the nodes are Interactive, and some are Inquiry, different tests needed to be performed on different nodes. Before the test plan was distributed to the testers, the nodes were updated according to which tests needed to be performed. "N/A" or "Not Applicable" was entered into cells for which that column's test need not be performed. For example, no need to check Required Fields of Inquiry Node "View Payment".

Financial System	**TESTS**					
	Required Fields	Drop Downs	Tab Sequence	Lookups	More Tests	Test Complete
• Payments						
• *Create Payment (Interactive Node)*					See Doc2	
• *Confirm (Action)*	N/A	N/A	N/A	N/A	See Doc2	
• *Cancel Payment (Interactive Node)*					See Doc2	
• *View Payment (Inquiry Node)*	N/A	N/A			See Doc2	

C) Updating Nightmare

Testers received a printout of the excel spreadsheet to follow as a test plan. They indicated test completion on the test plan by writing a checkmark in the test cells that were completed. These checkmarks had to be then typed onto the online document to indicate what was complete in each node.

Also, a defect that prevented full testing of a node was also written on the test plans by the testers, then transferred online. This was so that the tester would know to not revisit testing of that node until the preventative defect was closed.

D) Updating Module listing in Defect Tracking Database

Not only were the modules, nodes, and actions entered into an excel spreadsheet, but they were also entered into the defect tracking database. This created a DropDown list for the module field of the defect when the tester was entering a defect. This module could later be used in a search to find all defects of similar modules, nodes, and actions.

Every time the Node names changed, or Nodes were deleted or added, the defect tracking database also had to be updated separately in addition to the excel spreadsheet.

E) Time for Enhancements

This was not a finished project at the time of initial testing! Nodes were being added, deleted, renamed, and moved throughout the whole process. This meant adding new rows to the excel spreadsheet, shading nodes that had been removed, renaming nodes with new names, etc. Occasionally, nodes that were removed were returned, and this too had to be indicated on the test plan.

Changes to nodes and structure were emailed from programming lead to testing lead. The changes were then applied to the excel spreadsheet, and also to the defect tracking database. For example, if a node name changed or was moved to a different module, the defect tracking module DropDown list had to accurately reflect the new name or new structural placement.

F) Drawbacks

- Transferring testers' handwritten checkboxes from test plan to online document took unnecessary time, and there was room for error.
- Had to photocopy testers' test plans so they could continue testing while test plans were updated online.
- The online documentation never fully reflected current completion of work because testers were updating handwritten test plans while testing lead was making online updates.
- Because there were 2 test plans for the application (due to too many tests for one document, as earlier mentioned), preventative defects had to be updated on 2 different test plans, which is a duplicate effort.
- The enhancements and node changes were first written in email by the programming lead, then transferred online to the test plan by the testing lead, which is indirect, timely, and prone to error.
- Enhancements and node changes had to be made to the excel spreadsheet and the defect tracking database, which is a duplicate effort.
- Node additions, deletions, updates, and moves were not always accurately conveyed to the testing lead. For this reason, the testing lead continually printed the structure of the application, including modules, nodes, and actions, and compared the enhanced structure to the test plans, then updated the test plans accordingly. Talk about tedious!

Return to Table of Contents

V) The Database Solution!

Instead of this long drawn-out manual effort of test plan creation and updates, and enhancement updates, a database and GUI front-end can be created to maintain testing of generic functional testing across similar nodes.

A) Node Test PopUp

The following are examples of Node PopUps the testers can see while interacting with the database through the GUI front-end. The PopUp is interactive, and the tester can indicate which tests are completed, with audit information (tester and date) being stored in the backend, or front-end if desired.

Example of Incomplete Interactive Node:

Node Name: "Payments – Create Payment"

Type of Node: *Interactive*

Tests for Node:

Tests	Completed
1) Required Fields	√
2) DropDowns	√
3) Tab Sequence	
4) LookUps	
5) RadioButtons	√
6) Color	√
7) Font	
8) Add New Record	

Defects preventing test completion: #306 (OPEN)

*******Node incomplete********

Example of Completed Inquiry Node:

Node Name: "Payments – View Payment"

Type of Node: *Inquiry*

Tests for Node:

Tests	Completed
3) Tab Sequence	√
4) LookUps	√
6) Color	√
7) Font	√

Defects preventing test completion: (none)

********NODE COMPLETE!*******

Return to Table of Contents


B) Front-End Views

Besides the Node PopUp for testers, there can be many views of this database. For example, a view by Module or Node of percent completion, a view of Nodes that are pending defect fixes or information requests, a view of Module/Node structure to which the program lead can make enhancement changes, and many more views. These views can be seen online or in report format.

C) Benefits

- The test plan database will accurately reflect what is currently complete! As soon as the tester updates a test in a node, that information can be made accessible and accurate to any interested party via the form of report printout or GUI view.

- There is less dependency on testing lead to update online test plans with testers' completion updates and program lead's enhancements. If the testing lead is absent, the show will go on! Just look in the database for open testing items.

- The testing lead can be testing instead of typing in excel all day! This database solution minimizes the administrative testing role on the project, freeing up resources to help with actual testing!

- Previously the test plans had to be manually updated with color-coding to indicate which nodes needed data, which nodes' retesting was pending a defect fix, which nodes needed information before further testing could be completed, etc. With the current Database solution, each node can be flagged with such criteria, so that a reports can be generated by node or by category (ex: need data in order to test) for testing needs.

- The test plan database can be connected to the defect tracking database. Therefore, the defects that are preventing testing, as indicated by the tester in the node Pop-up, can be automatically flagged in the defect tracking database as high priority. This way those defects can be fixed first (example, blow-up defect #306 can be fixed before fixing other cosmetic defects, so testing can continue sooner). The defect tracking database can communicate to the test plan database, indicating when high priority defects are fixed, and node retest can occur.

- Tests can be indicated by red color in Node PopUp as high priority. Then tester will know to focus on those tests in the node, then move on to the high priority tests in the next node, and so on. Tests can also be added or deleted in background of database. These changes will automatically reflect in the node PopUp.

- If defect tracking database is connected to test plan database, defect tracking database can be scanned for keywords of defects (ex: "Problem with Checkboxes"), and keywords with high defects associated can be fed into the test plan database as new tests (ex: "Verify Checkboxes").

- Testers' performance can be tracked efficiently through audit trails of node testing completion. This can be used for evaluation purposes of testers, or for estimating remaining test effort on the project.

- As enhancements are made, node additions, deletions, name changes, or moves can be directly made by program lead into the database either through a view of the nodes themselves or a view of the entire module structure. Program lead could also indicate if a node changes type, for example, from Interactive to Inquiry, which would automatically change the appropriate tests in the background (ex: don't need to test required fields in an Inquiry node, so that test would be removed from the node).

  View of module/node structure as seen by program lead:

Financial Application
• Payments
• *Create Payment*
• *Confirm*
• *Cancel Payment*
• *View Payment*
• Receipts
• *Create Receipt*
• *Confirm*
• *Cancel Receipt*
• *View Receipt*
• Reports
• *Payment Report*
• *Receipt Report*
• *Summary Report*
• Maintenance
• *Edit Accounts*
• *Edit Currency*
• *Add Users*
• *Set System Date*
• *Set*

VI) Closing

Hopefully this journey has offered you the assurance of the efficiency you can obtain through the implementation of a Test Plan Database. Don't be the database – let technology assist you! I invite you to discover the benefits of this solution for yourself and enjoy the time and freedom you will gain!