



PACIFIC NW
28TH ANNUAL
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010



ACHIEVING
QUALITY
IN A COMPLEX
ENVIRONMENT

*Conference Paper Excerpt
from the*
CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

Incorporating user scenarios in test Design

April Ritscher
Senior Test Engineer
Microsoft Corp.
aprilri@microsoft.com

Abstract

How many of us have tested an application and certified that it met the requirements stated in the functional specification, only to find out that it does not meet the business need? Many times as software test engineers we are brought on to the project after the requirements have been gathered. This gives us very little visibility into the early discussions on what the user needs to accomplish with an application solution.

As part of our work to improve the software testing process for our group (LCAIT), we have been looking into ways to move upstream in the process and ensure that we add business value by testing our applications based on user scenarios that reflect the usage of the application. This required us to change our approach in writing test cases and changing our focus from the traditional functional testing to user scenario focused testing. To achieve that objective we used visual representations of possible user actions and application responses that align with the user scenarios.

This paper will describe how we break out our functionality into individual test cases and use these as building blocks to test the end to end user scenarios. It will also describe how we extract information from the flowcharts to be used during manual and automated testing.

Biography

April Ritscher has been a senior test engineer at Microsoft for the last 10 years. As test lead, she has worked on a variety of different projects including a project for Bill Gates. She has received several awards including 5 Ship-it awards and 3 IT Pro awards. She has worked on several global teams with both India and China. She is very passionate about quality and always looking for ways to improve the software testing process.

1. Introduction

As test engineers we are asked to ensure the quality objectives of an application are met. In the past we have accomplished this by comparing the functionality developed against the functional spec and certifying that it matched. Now if you asked someone in IT how we feel we have delivered against customer expectations you would hear that we feel we meet their expectations 94% of the time. Now if you ask our customers, they would say we meet their expectations 48% of the time. How can this be? If we certified that our application met the functional spec how can we have such a large discrepancy between our assessment and that of our customers?

The problem we have found is that the customers assess the application based on whether it allows them to accomplish their day to day activities efficiently and easily regardless of what was defined in the functional specification.

What we found is that using the Functional specification or Requirements as your measuring device does not always ensure that the application will meet the user's needs. Instead we need to move upstream in the process and find out what the customer needs to be able to accomplish in an application agnostic way. The best way to do this is to gather user scenarios at the beginning of a project and incorporate these all the way through the project life cycle.

2. Goals

In order to shift from the mind set of being a test team that merely certifies an application to a Quality assurance team that influences the design means that we need to move upstream in the process. One of the main problems that we have in test is that we get involved at the end of a project. According to Authors Roger S. Pressman and Robert B. Grady the cost to fix software defects varies according to how far along you are in the cycle. As you can see from the figure below, issues found later in the project are significantly more expensive than those found in earlier phases.

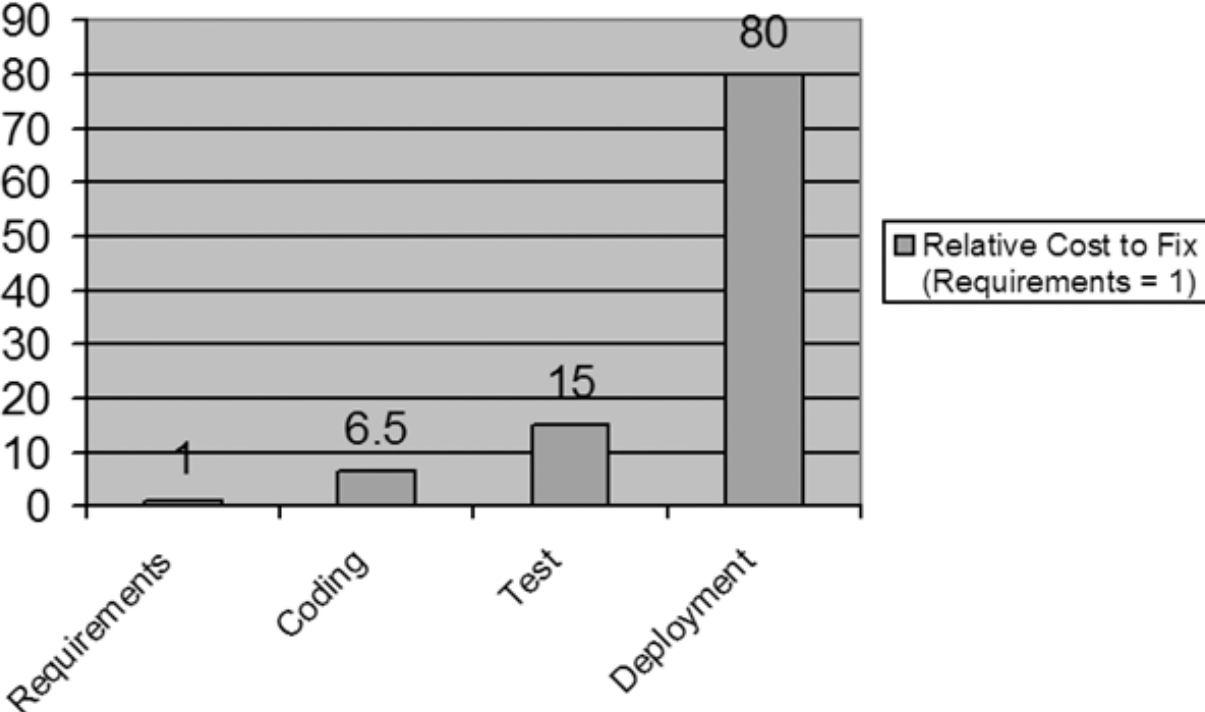


Figure 1: Variations in costs to fix a software defect

By involving the test team earlier in the project we are able to increase our capacity as well as become more influential on application design.

Another way to increase the capacity of the test team is through Automation. One of the major deterrents to automation is the cost. One of the goals we had during this approach was to find ways to reduce the amount of maintenance cost needed on our automation code.

Finally our overall goal of our process was to ensure that any solution we delivered met the business needs and that we didn't end up in a situation where we coded per the specification but didn't meet the user expectations.

3. What is a User Scenario?

The first question you may ask is what is a user scenario? A scenario is a concrete narrative story told from the customer's point of view that explains their situation and what they want to achieve. A well written scenario will be application agnostic and will not say how they will accomplish their goals but will focus on what those goals are. It should also describe the customer "dream state" of what they would like to be able to do, that they may not be able to do today with existing tools or applications. A scenario is real world descriptions which often span more than one feature.

Example of a scenario:

Jane is at the airport at 9am waiting for her flight to Chicago for a business trip, leaving at 10:30am. While waiting for her flight, Jane notices an important message from her colleague. The presentation in Chicago has been moved to a new location, and the partner has asked to see 3rd quarter projections as well as 2nd quarter. Jane responds to her colleague to let her know that she got the message but doesn't have the 3rd quarter projections on her computer – can Sue send it to her immediately, before her flight leaves? A couple of minutes later, Jane gets the 3rd quarter projections and has just enough time to review them before her flight. She is relieved and confident that she will be fully prepared for her meeting.

4. Benefits of testing with user scenarios

How do Scenarios help us? A well written scenario allows you to see your solution through the user's eyes and should paint a vivid picture of their user experience. It also helps to identify which features will be critical to the user's day to day business process and reduce the risk that these could be inadvertently dropped out of scope. They also help to identify design gaps earlier in the design phase. By comparing your application design to your user scenarios you can easily see whether the proposed solution will help or hamper their ability to accomplish their goals.

Using scenarios during the test phase also helps you to prioritize your defects. If a bug is found that prevents the user from accomplishing their goals laid out in the scenario this becomes higher priority than other bugs that may not be part of the user's main workflow.

5. From Conventional To Innovative

In order to incorporate user scenarios in to our testing we needed to shift our mind set on the definition and representation of a test case. Typically in the past we had created functional test cases which were text representations that included all the steps you needed to execute in order to test a certain feature. The problem with this is that we ended up looking at functionality in isolation as opposed to looking at how the user would use the application to accomplish their goals. Also we needed to have thousands of these functional test cases to completely test all of the application features. Reviewing test cases can be a mind numbing exercise and if you have thousands of them, the chances of getting meaningful feedback from your reviews is slim to none.

Instead we decided to visually represent our test cases in Visio in order to facilitate reviews and ensure that our test cases aligned with the user scenarios. Why Visio? We chose to use Visio because it was readily available to the entire team since everyone already had office installed. We also needed a tool that would allow us to export our workflows in a textual form to be integrated with our test case management tool. However which tool you use is not as important as the fact that you visually represent your workflows.

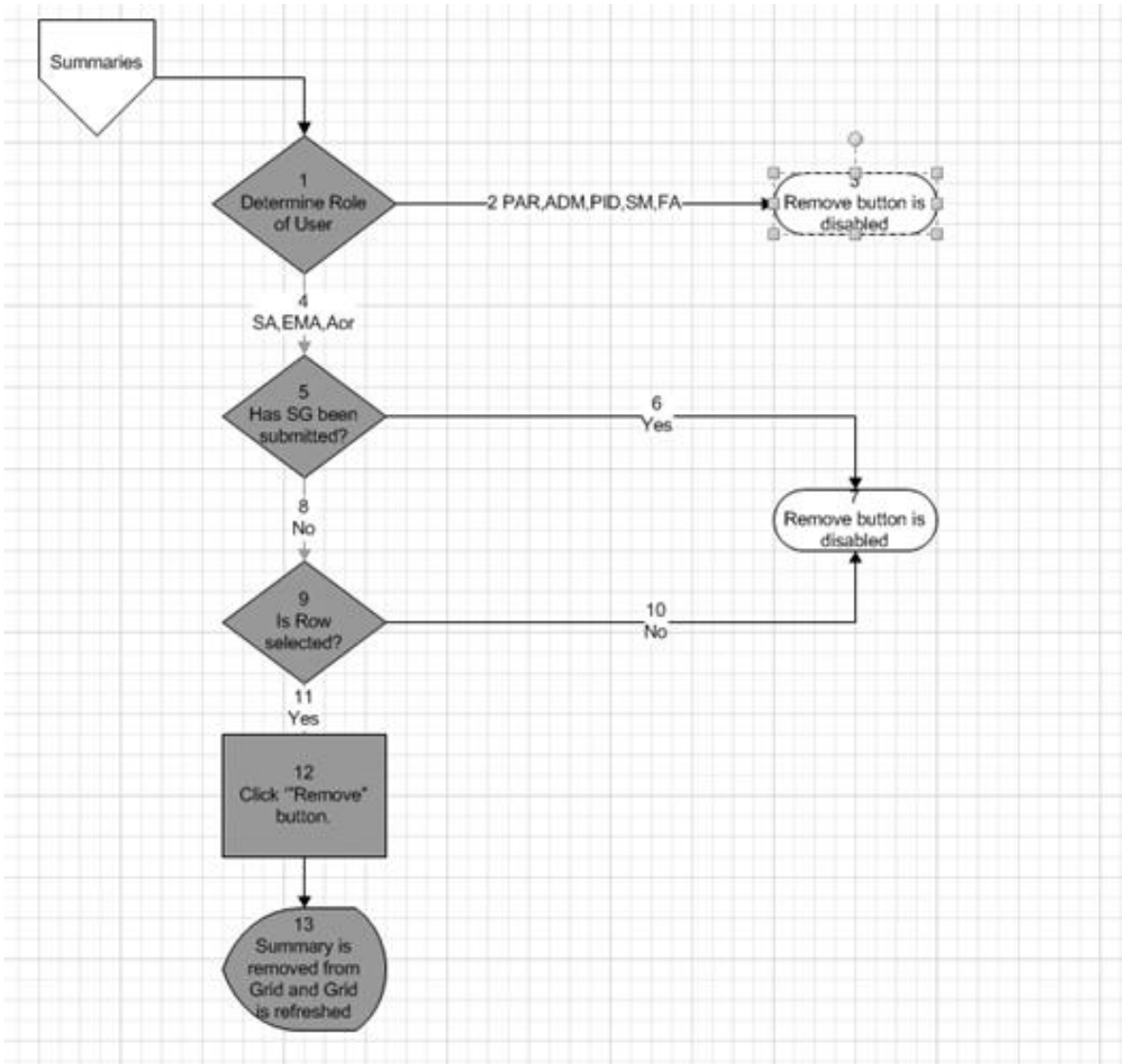


Figure 2: An Example of a workflow in Visio

Another area of focus we had was on automation. One of the challenges we had in the past was that our automation was framework dependent and any decision to move to a different automation framework would require an additional automation cost. Instead we transitioned to a model using C# that allows us to use any framework we chose as long as the appropriate classes are public.

Finally our last area of focus was on the way we incorporated data into our automation testing. Typically in the past, if we had multiple data variations for a given test case, we duplicated this test cases multiple times. The problem with this is that there was a lot of redundant data which resulted in a much higher maintenance cost. Instead we extracted the data variations in to their own entity that allows us to execute a single test case against multiple variations. In our line of business most of the business logic is very data dependent. By extracting this information we were able to reduce the number of test cases by 88%.

You can see from the figure below how the three areas of focus map to our original goals.

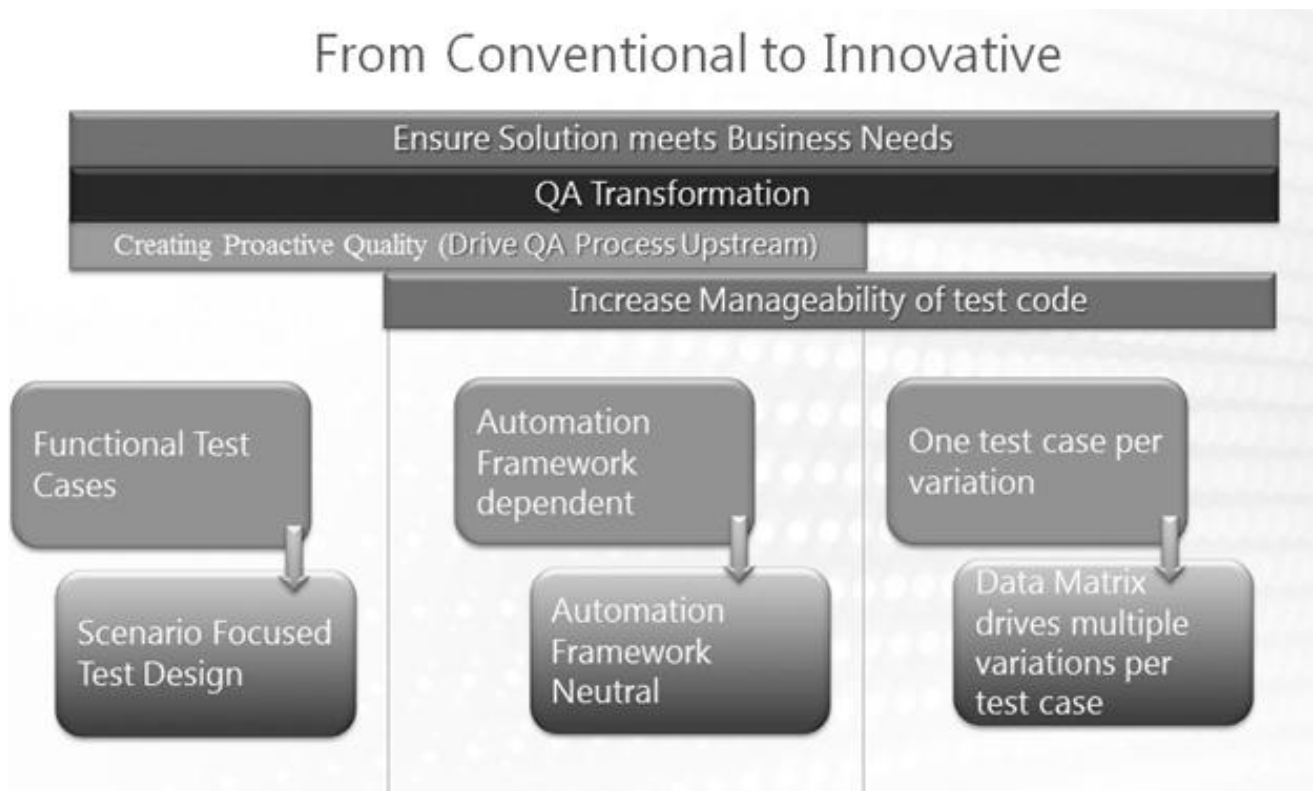


Figure 3: Aligning our focus areas to our Goals

6. Scenario Focused Test Design:

How do we pull it all together? In the figure below you can see how we incorporated user scenarios in to our testing process.

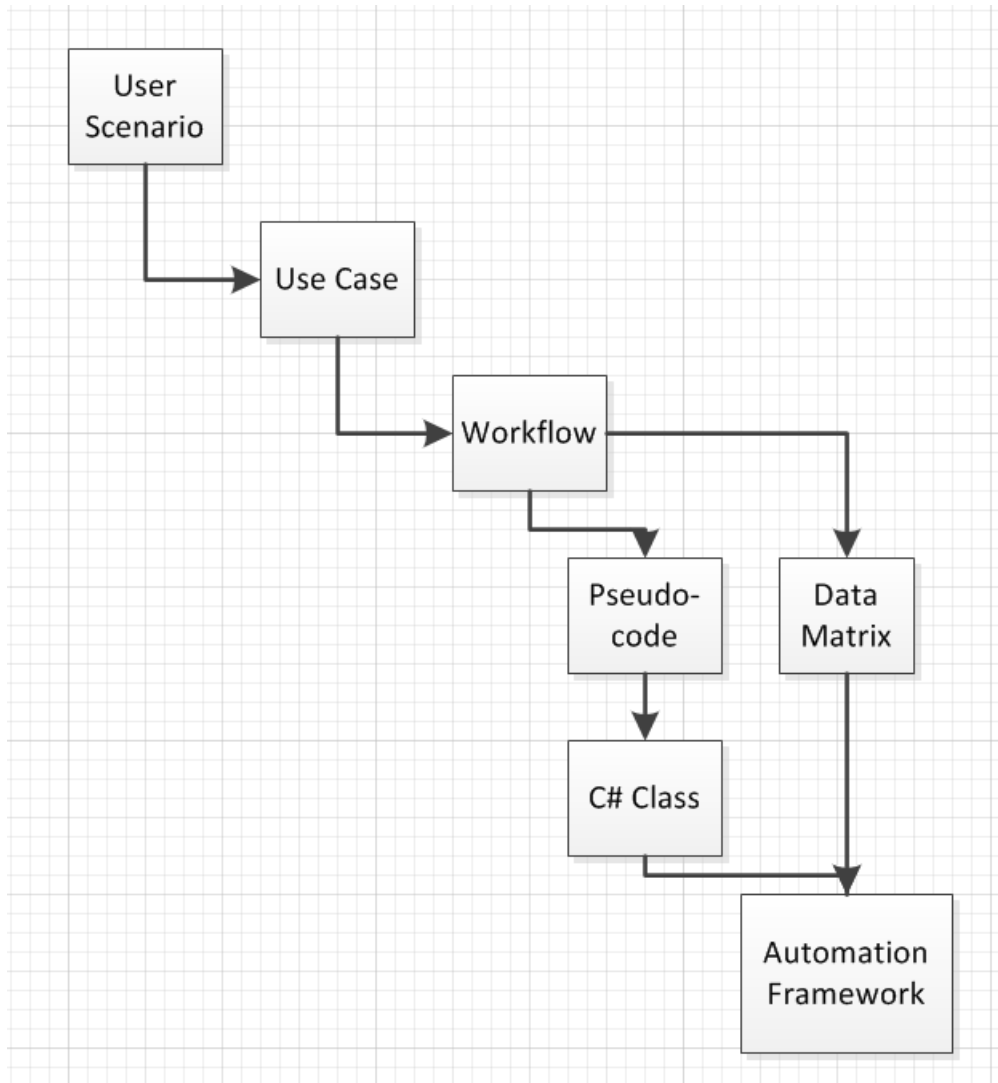


Figure 4: User scenario focused testing workflow

User Scenario: We start with the user scenario which describes what the user wants to be able to accomplish.

Use Case: The User scenario is then translated in to a high level flow in Visio that corresponds to a use case

Workflow: We then take the use case and expand on it including information from the functional spec on how the application is expected to respond the user actions. We document their main flow as well as any alternate flows that they may need to accomplish.

We use tabs in Visio to compartmentalize our functionality into manageable functions that can be strung together to align with a variety of different user scenarios. We simplified the flowchart process by using only 6 basic shapes in Visio. The first type of shape is a user step that defines what the user will do. The second shape is an application response. The third shape is a decision that represents any time there is a possible branch in the workflow. The fourth shape is a dynamic connector which is used to tie the shapes together. The fifth shape is an off-page connector which is used to tie the different tabs together. The final shape is a terminator which indicates when the user has reached the end of a particular function and should be returned to the originating function.

Using built in functionality in Visio we associate different meta-data to these shapes. This includes whether this step represents the main or alternate flow, any conditional hints like the beginning of an IF\Else block and the function name that corresponds to the tab name.

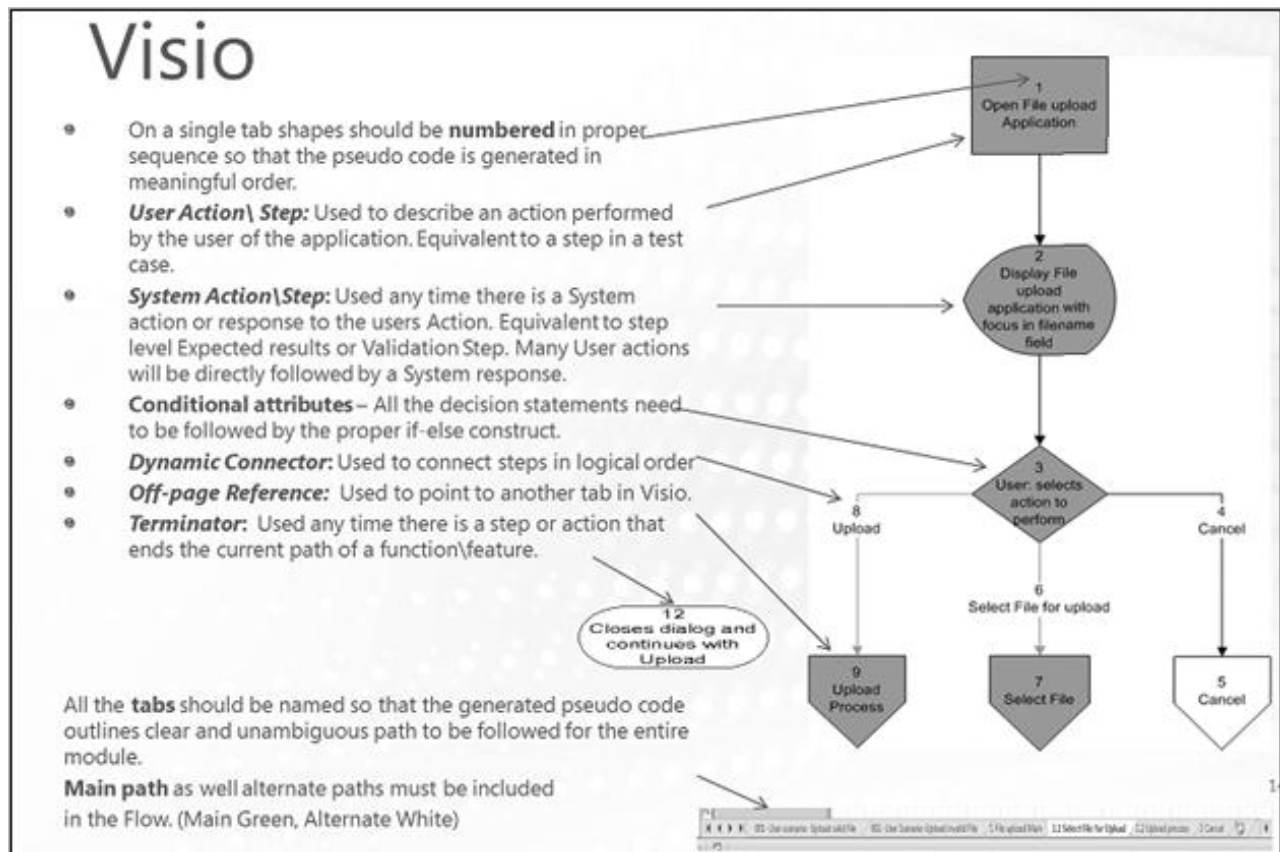


Figure 5: Shapes in Visio

Data Matrix: Using the visual format of the test cases makes it easy to identify where different data may be required in order to navigate through all of the possible workflows. We document these different data variations using XML.

Pseudo-code: Using built in functionality in Visio, we extract the data from the workflow into pseudo-code. The pseudo code reads as steps in a test case that we import in to our test case management tool. The pseudo-code also includes the embedded meta-data such as IF\Else hints or Go to statements.

C# Class: Our automation engineers translate the pseudo-code in to C#. Since the pseudo-code is an extraction of the workflow it contains all of the business logic from the workflow so a deep functional understanding of the application is not required to translate into C#.

Automation Framework: We execute the C# classes along with the data matrices in our automation framework. Since the classes are written in C# you could use a variety of different automation frameworks.

7. Conclusions:

By incorporating these changes in to our testing strategy we have ensured that our goal of meeting the business needs remains our top priority. We have adopted these processes on 2 project releases in my group so far. One project was released in to production in January and the other project releases in September of 2010.

7.1 Lessons for Others

1. Use User scenarios as a base for all testing

Our user's satisfaction has been much higher on these projects than on projects in the past and this has increased their confidence on our ability to deliver to their expectations

2. Visually represent your test cases

By presenting our test cases in Visio we made it much easier for other disciplines to review and provide feedback on test cases. In my ten years at Microsoft this has been the first project that we have had 100% review coverage of our test cases. This resulted in much higher quality test cases than we have seen in the past.

Also due to the quality of the pseudo-code being generated we have been able to reduce the skill requirements of our automation engineers and have saved over 150k this year(This is based on the cost per hour difference between vendor levels multiplied by the number of hour spent on the project)

3. Move QA Process Upstream

By involving our team earlier in the project we have been able to identify bugs before receiving an official drop from development. This has allowed us to test much deeper functionality that would have previously been blocked. On average we identified about 10 bugs per application build that fall in this category. So far on the project this has been a savings of 180k. (We spend about a thousand per bug fix and have 18 drops scheduled for the current project. This will be a reduction of over 180 bugs)

4. Extract data variations in to data Matrix

We have also seen a dramatic cost savings in the areas of automation. By extracting our data matrix in to a separate entity we have reduced the number of hours required to maintain automated test cases by 4000 hours.(We reduced the number of test cases from 1048 to 128 and on average we spend about 4 hours per project per test case on maintenance)

8. Acknowledgments

For their valuable feedback and review:
Hisham Gaber
Yogesh Kulkarni