A discussion on testing levels & bug regression

Abstract: The goal of this paper is to discuss the levels of our test efforts and bug regression and their outcomes within the span of a project. We all do different levels of testing as developers and as testers in any given project. Developers are better at unit testing as they are closely familiar with their code and testers are at system testing as they are better at designing the test settings. Test settings are usually custom designed and fairly unique from place to place. I will limit the discussion specific to our design.

Test & Fix: In order to keep our discussion brief and simple, I will not go into the detail of our development environment, but will simply state that ours is a Quasi-Agile environment. We build our application by working on multiple components at the same time over a series of repetitive process. We have a fairly good documentation of the user stories and design specs, but they do change throughout the course of development. As developers are writing code, doing code reviews and performing unit tests, we are creating test scenarios, reviewing test scenarios (time permitting), writing test cases and reviewing test cases (very rarely). In Test & Fix, we will be testing primarily the positive test scenarios (happy path). Here is an example,

	Test Coverage	
User stories	**Description**	**Test Scenarios**
DC4.1.4.1.1	DC console being non-modal will appear in every page within a probe	DC console will appear in a problem with, - An empty page - Multiple empty pages - Maximum number of empty pages - A page with an application - A page with multiple applications - Multiple pages with multiple applications DC console will not disappear when, - Page layouts are changed - Applications are added and deleted within each page layout - Applications are swapped within each page layout - A program or function is opened - A program or function is viewed - A program or function is edited - A program or function is being executed

		- Pages are dragged and dropped in tree view - Pages are copied and pasted in tree view - Pages are copied and pasted in page sorter view DC console will disappear when, - A new problem is added - Existing problem is deleted
PE3.1.3.1.1a **PE3.1.3.1.1b**	Program Editor is a menu item Program is created, opened, viewed and edited through Program Editor	

For the user story DC4.1.4.1.1, we will only test the following test scenario
DC console will appear in a problem with,

- An empty page
- Multiple empty pages
- Maximum number of empty pages
- A page with an application
- A page with multiple applications
- Multiple pages with multiple applications

We will probably not test the subsequent test scenarios from the metric for simple reasons like, code for Program Editor may not be in yet or, we don't know the rules of interactions between DC console and Program Editor or DC console and change of page layouts. And so, it makes sense to have a shorter test cycle for Test & Fix. As all the different components are built and get integrated, we are ready to start the next phase.

Integration: Since bulk of the changes is out of the way, we now have almost a static target to take an aim at. It is very difficult to shoot down a moving target when user stories are getting defined and redefined and code is constantly changing to reflect the user stories. In some cases, new user stories are added and some existing ones are removed. At this phase of testing, expectations are such that we are pretty much done with all the changes. But, we all know that, it is not always the case. Our hope is that there will be no new user stories and a very few new definitions of any existing one at this point. De-scoping of a user story is fine as it doesn't require any new work and rework from our part. As the target is fairly immobile at this stage, we are now poise to attack. For the same user story (DC4.1.4.1.1), we will test the following test scenarios,
DC console will not disappear when,

- Page layouts are changed
- Applications are added and deleted within each page layout
- A program or function is opened
- A program or function is viewed
- A program or function s edited

- A program or function is being executed
- Pages are dragged and dropped in tree view
- Pages are copied and pasted in tree view
- Pages are copied and pasted in page sorter view

DC console will disappear when,

- A new problem is added
- Existing problem is deleted

Since a significantly large number of test cases are executed during integration, this is perhaps the longest of all test cycles. As a result, a large number of fairly complex and difficult bugs are generated during this test cycle. We don't expect the recurrence of the any existing bug (bugs found in Test & Fix). As integration nears the end, we see the bug reporting frenzy flattens out gradually. This is the part that worries me the most. The reason we are not finding nearly as many bugs as we did earlier tells me couple of things. We have either found most of the bugs or, we simply ran out of ways to find more bugs. We always like to improve on the depth and breadth of our test coverage. But, do we know which way to go? Do we go right or left to add the breadth or up or down to add the depth? Do we know how far left or right and up or down we need to go? There is no easy answer to these very difficult questions. However, the answer to some of these questions just might be in bug regression. Here is how it works,

Bug regression: Once we log a new bug and triage it to make sure that it is indeed a bug, we then a write a test case (if we don't have one already) to address the bug. But we need to do more; we need to,

1). Regress around the bug: We need to create a set of test scenarios around the bug and write a number of test cases based on those test scenarios. So, next time when we verify the bug, we will also execute the set of test cases associated with the bug. Let's take this bug (a true bug) for instance,

Bug Summary: DC console disappears when opening a program or function in Program Editor

In this case, we already have a test case based on our test scenarios from the metric.

Test scenario: DC console will not disappear when a program or function is opened.

As a matter of fact, we found the bug by executing the very test case that illustrates the test scenario. We also know the cause of the bug from the bug tracking tool. It turns out that, the function GoToWidgetOnPage() does not handle moving console between pages very well. The solution to this bug was to add code to this function to better handle the moving console between pages. In order for us to regress this bug, we need to run test cases with the following test scenarios (see Test Coverage metric),

DC console will not disappear when,

- Page layouts are changed
- Applications are added and deleted within each page layout
- Applications are swapped within each page layout
- A program or function is opened
- A program or function s viewed
- A program or function s edited
- A program or function is being executed
- Pages are dragged and dropped in tree view
- Pages are copied and pasted in tree view
- Pages are copied and pasted in page sorter view

Since we already have test cases, we don't need to create any new test scenarios or write new test cases in this case. The current test cases provide a good coverage for this bug.

2). Follow the code: Since changes are made to function GoToWidgetOnPage(), we need to know if we are

using the same function in any other part of the application. The premise of going after other areas of the application has to do with the notion that they are guilty by association. This information has to come from the developers. We will track this information through bug tracking tool by configuring the tool such that the effected areas (GUI and functional) are documented in the tool. Once we have this information, we will either use the existing test cases (if we have any) or create new ones or do both to test the areas that use this function. As a result, next time when we verify this bug as part of our regression, we will also test the parts of the application guilty of using the same code and thus making sure that fixing this bug does not cause any unpleasant occurrences in any other region of the application. It is possible that some bug fixes are strictly isolated and have no effect on any other areas. But for the bugs with shared code (most are, since codes are – reusable), this approach of bug regression will significantly increase the depth and width of our test coverage in all directions.

Exploratory: We usually open our exploratory sessions soon after the integration. This form of unscripted, unstructured and anything goes type of testing is actually very refreshing. The only scripted part is a small document with a few lines of text which explains the areas of the application are to be tested. Exploratory sessions are like Easter egg hunting; a race to find as many bugs as possible with as many participants as possible. Sometimes very strange bugs come out of these exploratory sessions. Here is one (a true bug),
Bug summary: Cursor position changes
Bug description: - Open the application
- Create a program in Program Editor
- Enter a line of text
- Go to the end of the line by pressing the right arrow on the keyboard
- Do a mouse click on the current cursor position

Result: Cursor jumps to a different position.
This is probably a very minor issue and the chances are the user will always use the mouse instead of the keyboard to change the cursor position. Since this is a bug, we may want to create a couple of more test scenarios around it by,

- Going to the beginning of the line by pressing the left arrow on the keyboard
- Going to the middle of the line by pressing the right arrow on the keyboard

It is comforting to know that we usually don't find too many show stoppers from the exploratory sessions. At this stage of test cycle we don't anticipate any critical issue but, we do find one or two every now and then.

Currently the second part of bug regression (Follow the code) is not part of our test efforts as it requires configuring the bug tracking tool. It also requires participation from the development and thus needs a coherent effort from both testers and developers for putting this in place. This will take some time. But once in place, it will benefit both QA and development and the outcome will be a better product.