# LEAVE NO STONES UNTURNED

By Mihir Kamdar

Every test manager's nightmare is finding out that not enough testing has been done. And the paradox lies in the fact that enough testing often comes with a heavy tag in terms of cost and efforts. If you under-test the product due to time and cost limitations, eventually, the cost of support and maintenance will increase. The real challenge is to get the optimum quality at the lowest cost.

Let us understand the problem little bit in depth. Let's consider the case of a web-based product for which around 1000 test cases have been written. Let us assume that this product is supported on 2 database servers, 3 Operating Systems, 2 web/app servers and 2 web browsers. This makes a total of 24 combinations of the supported platforms.

Executing 1000 testcases on each of these combinations would give any test manager a shot of confidence about the quality of the product by having looked at every nook and corner of the product. But that would cost efforts of around 22 man-months (assuming an average tester can execute around 50 testcases a day)!! This is too high and practically impossible. It would further delay the time to hit the market.

The first thing that comes to mind is Automation. Automation can surely solve the problem. But, creating the automated testing environment will also require time and efforts. Also, if the product under test is not stable, automation will require lots of maintenance efforts. So, it is not practical to employ automation for such products. Once the product is stable, automation is a must for regression testing. For the products that are not yet stable, the only option remaining is to take a risk, and in a calculated way. How?

First things first. The manager has to lay down the goals for the exercise. The goal should be to discover ALMOST ALL the defects within the permissible time & cost limits. Here is an approach to resolve the problem.

- **Know the features of the application under test well**. This will help you to identify the dependency of each and every feature with the platform under test. For example, if a particular web-page is static in nature, you do not require testing it for different databases or OS. You can choose to test it for different web browsers though as different browsers may render such pages differently. You need to know what component of the supported platforms would have affect what feature.

- **Know the supported components and their intricacies well**. This would lead towards identifying the probable defects due to change in the platform component. For example, if Windows XP & Red Hat Linux are two supported OS flavors, knowing the difference between the two OS would be beneficial to identify how these differences can affect the product under test. Yours tests can be aligned towards looking out how these changes affect your system under test. For eg. one difference between these two OS is that the file names are case sensitive in Linux, while they are not in Windows. Let's say that your product uploads files to the server running one of these OS, you can check for windows that if you upload the same file by changing the case of the file name, it would either overwrite or prompt for duplication. In Linux, it would simply store it as another file.

- **Come up with the list of the most important platforms**. This means that you will require knowing what the most popular platforms with your customers are. Even the least severe defect visible to everyone will be more critical than the most severe defect that is visible to only few customers. So, it is important to test maximum on the most popular platforms. If you test the maximum on the most popular platforms, most customers will be satisfied with the quality of the product, meaning less support required.

- **Wear your thinking hat** and decide what feature need not be tested or need not be fully tested on what component. It would be wise to include the developers and other stake holders in this crucial decision making process. It is in this stage where you are identifying the risks – the calculated risks. Depending on a variety of factors like the stability of the product, the kind of defect prevention measures followed during development and some historical data about the defect counts in such kind of projects in your team, you can make a decision on how much repetition of tests is necessary. For example, if the product is not very stable, you might want to test it more. If the project is about enhancing an already stable product, you might just want to test it once or twice before releasing it. Knowing the kind of defect prevention measures followed during development and the historical defect trends would enable you to forecast the number of defects that can be uncovered. This will help you to identify how much testing is necessary. More number of defects means more testing is required before declaring a product eligible for release.

- **Risk and Mitigation Plan**. It is also important to identify the risks and have a mitigation plan ready at this stage. The risks can be what if more defects than planned are found, or what might happen if a particular component is under-tested on a particular platform. It becomes easy to choose what risks you want to take before going ahead, and re-plan your strategies if required.

- Based on the decisions taken on what to test and what not to test, **come up with a matrix** which lists down all the supported platforms on one axis and the features on the other axis and mark what features NEEDS to be tested (fully or partially) on which platforms.

Using this approach, you can significantly reduce the testing time and cost, and minimize the risk of leaving many stones unturned.

---

The author is a QA Lead at eInfochips, Ahmedabad (India). The author has an experience of over 6 years in system testing and development. The author can be reached at mihir.kamdar@gmail.com