Creating Compelling Checklists

Mindfire Solutions

www.mindfiresolutions.com

May 12, 2003

Abstract:

This paper propels the goodness of custom-made checklists and aims to help test organizations build their own checklists.

Introduction

How often do you browse the Internet in order to get hold of a perfect checklist for your testing requirements?

How often do you wish, if someone somewhere in the world had prepared a checklist that you could directly apply to testing your product?

How often do you get disappointed that all the checklists floating on the web are either non-extensive or overly exhaustive?

How often did you wish, had you recorded repeatable test skills from your previous project that can be used over and over again for similar projects?

If your answer to these questions is 'ALWAYS', then it's time you thought about preparing your own testing-checklist.

All of us in testing business would fairly agree to the advantages of checklist driven testing. Checklists are handy tools, which not only eliminate the probabilities of missing important and repeated tests, but also help systematizing tester's job. Moreover checklists are lifesavers when you make use of a pre-release checklist, while your company intends to make release in haste. Checklists add value to testing in more ways than you can actually think of. (Ref: "Checklists=Overview.doc")

Why is a tailor-made checklist better?

While testing you not only work with the software, but also with people, procedures and other managerial aspects related to a specific organization. This probably answers why a checklist made in some other organization may not completely suit to your testing needs.

You know the mistakes your developers are apt to make. You know the tests your testers may forget. You know the ingenious ideas implemented to discover critical bugs. If you make a checklist by considering these attributes, you may end up making a very usable and helpful checklist.

Building custom checklists

Building a checklist is not that difficult. At Mindfire, we have built our own checklists by applying some basic procedures.

Step 1 - Start Gathering

The first step towards building checklists would be to gather more and more testing tricks as and when you find them.

Start with your current project. Make a simple Worksheet, where you will record the initial points. Make two columns, one for numbering and second for describing the point.

You are ready to start!!!!!

For every bug that you file, scrutinize your own activity. Analyze whether the bug is an irrelevant program error, or a substantive one. If you are convinced that the point can make it to your checklist, go ahead and make an entry.

Just to make sure that you don't end up making too many unnecessary entries, here are few checkpoints.

1. **Record Tricky Actions:** If you invested on some innovative ideas to discover a bug, make sure you record it. Don't let it go unnoticed; else you may forget to apply these tricky techniques elsewhere in future.
2. **Record apt-to-forget tests:** Many simple points that you intend to check, but forget every time because they are too simple to remember. How often do you remember to check tab sequence in a form? But every time you remember to test it, you can almost guarantee a disorder.
3. **Record repetitive bugs from development mistakes:** Development team makes repeated mistakes across projects. Record common weak areas pertaining to development across multiple projects.
4. **Record accidental bugs:** There will be bugs in the software, which you come across accidentally. You may have never thought about ways to discover those bugs, but luckily you found them while exploring through the software. Make sure that such bugs make an entry to your checklist. While testing a desktop-software, accidentally we discovered that changing the PC time zone results in garbage data in most date/time fields. Now it's a must test criteria for every desktop software that we test.

You will be surprised at the number you manage to gather at the end of week1.

Step 2 - Classify points

Once you have gathered a decent number of points for you checklist, the next immediate step should be to organize them.

1. ***Generic and Specific tests:*** The first classification should be to separate out generic tests from program-specific ones. Some bugs seem to be typical and

specific to the project. Others have probabilities of appearing in other programs too. The generic checklist will be more significant across projects. But do not throw away the specific tests. We will be talking about how to make use of program-specific checklists later.
(Ref: "Checklists=Organization.doc" for further classifications under Generic and Specific checklists)

2. ***Functional Grouping:*** Group the points based on their functional similarities and label them under different sections. E.g. for web application checklist, you could have following sections.
    a. Session Management
    b. User State Management
    c. Accessibility
    d. Usability
    e. Web page and data validation
    f. Performance
    g. UI (Broken links and page navigation etc.)
    h. Browser Compatibility
    i. Server Testing (Load, CPU/RAM Usage)
    j. Access Channel (Dial Up, Dedicated Connection)

Classification will make referencing easy. Also, when you add more points, you can directly place them under specific sections.

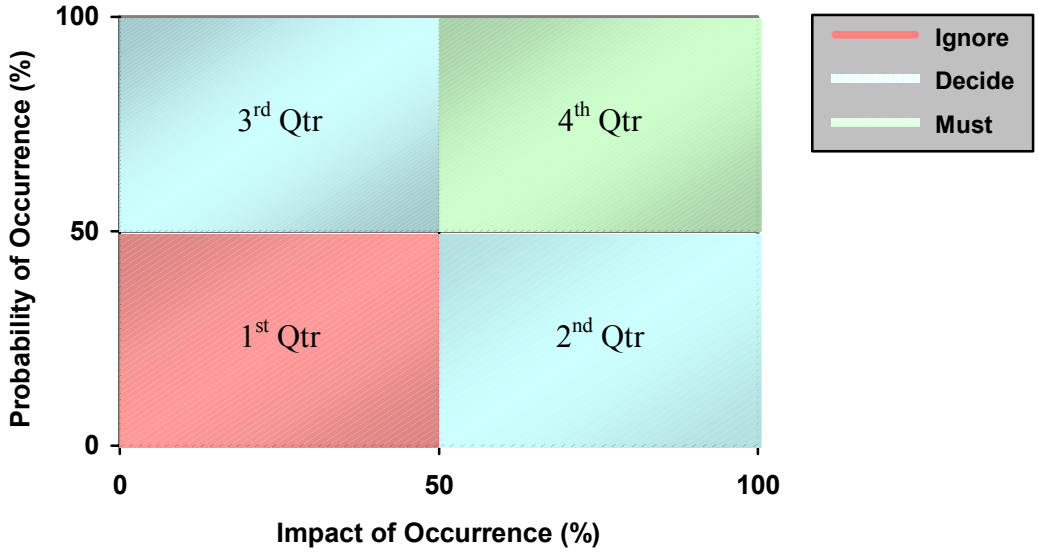
Step 3 – Eliminate - Deductive reasoning

The next step should be to cut down on simpler, redundant and unimportant entries. A checklist is effective when it's focused, substantive and most importantly brief.

The first two steps may result in too many entries in the checklist. You may not find it very sensible to collect entries first and then follow an elimination procedure. You would rather not enter these points at the first place.

But we suggest that you think about elimination, only once you are through the first two steps. The reason being, if you start eliminating from day1, you may end up ignoring a lot of important tests. Also it's easy to make choice from a bigger set of tests as it helps identifying important and high-priority tests over simpler ones.

The elimination criteria should mainly be based upon two important factors - ***Probability of Final Occurrence*** and ***Impact of Occurrence***. Revisit every entry in your checklist and evaluate against these factors. If the probability of existence of a bug is high for every build, it is a must entry in the checklist. So also, if the bug seems to occur in less instances but impact of the bug could be serious.



In the above graph, tests falling under the 1st quarter are neither frequent nor have serious impact. Hence these could be ignored as checklist entries. Tests, which are placed in 2nd and 3rd quarter, are high scorers in either of the two mentioned criteria. These need reconsideration before making an entry to checklists. Tests under 4th Quarter are the most effective ones and hence make definite entry to checklists.

Step 4 – Review, Evaluate, Optimize

The checklist is at its most effective when it's short and comprehensive; yet easily understandable and wide-ranging. Here's how the checklist can be optimized towards perfection.

1. **Understandable**: Get your checklist verified by other testers. Check if it's difficult for others to follow and implement tests directly from the list. All entries should be short and self-explanatory to be easily pursuable by other testers.
2. **Look outside your organization:** Refer to similar checklists created by others. Web is a good repository for referencing other works. Pick up tricky thoughts and missed points from such checklists. This way you can ensure a comprehensive checklist.

The key to make successful checklists is to constantly review, evaluate, update and upgrade.

Checkpoints…

Checklist should be evolving

Good checklists don't happen overnight. Or even if they could, we bet they won't be usable. Good checklists arise from practical experience; grow with time, projects and with more and more testing. Checklists mature and grow robust when there is serious implementation across projects.

Checklists should be small

Keep an eye on the size. An overloaded checklist not only deviates testers from applying the same, but also creates an illusive impression of being too theoretical. Keep minimal but important entries to make the checklist practically usable.

Add to your work routine

In your daily testing schedule, make notes of points that can enter the checklist. Update the points in the checklist at regular intervals. Similarly, application of checklist should be on a regular basis. This helps to check practical usage of checklist points and get rid of ambiguous and unnecessary entries. You can ensure an up-to-date checklist by enforcing regular usage and update.

Conclusion

Nurture a checklist culture that encourages learning from every bug and applies the lessons learned across multiple projects.

Mindfire Solutions is an IT and software services company in India, providing expert capabilities to the global market. Mindfire possesses experience in multiple platforms, and has built a strong track record of delivery. Our continued focus on fundamental strengths in computer science has led to a unique technology-led position in software services.

To explore the potential of working with Mindfire, please drop us an email at info@mindfiresolutions.com. We will be glad to talk with you.

To know more about Mindfire Solutions, please visit us on www.mindfiresolutions.com

---