Organizing Testing Checklists

Mindfire Solutions

www.mindfiresolutions.com

April 21, 2003

Abstract:

When Test Organizations plan serious implementation of checklist driven testing, systematic approach to deal with these rapidly growing custom checklists becomes a critical factor. It is not only difficult to manage growing number of checklists, but also successful implementation becomes questionable. This document describes how checklists can be further classified according to their usage and hence be organized.

Introduction

Before test organizations strategize to adopt checklist driven testing, they must invest towards collecting a wide variety of checklists. While realizing the goodness of custom-made checklists along with ready-to-use ones, test teams cannot ignore the fact that disorganization can lead to disastrous failure of the whole strategy. This document intends to help test professionals organize and manage checklists as they grow in size as well as in number.

Why is Systematization important?

Systematic organization of checklists becomes important as it attacks problems related to development, implementation and management of checklists.

Easy to Use

When checklists are classified based on common focus, it helps testers pick up suitable checklists quickly and easily. For example, while testing desktop applications built on VB and MS Access, the tester can pickup a VB Language-oriented checklist as well as an Installation checklist to start with.

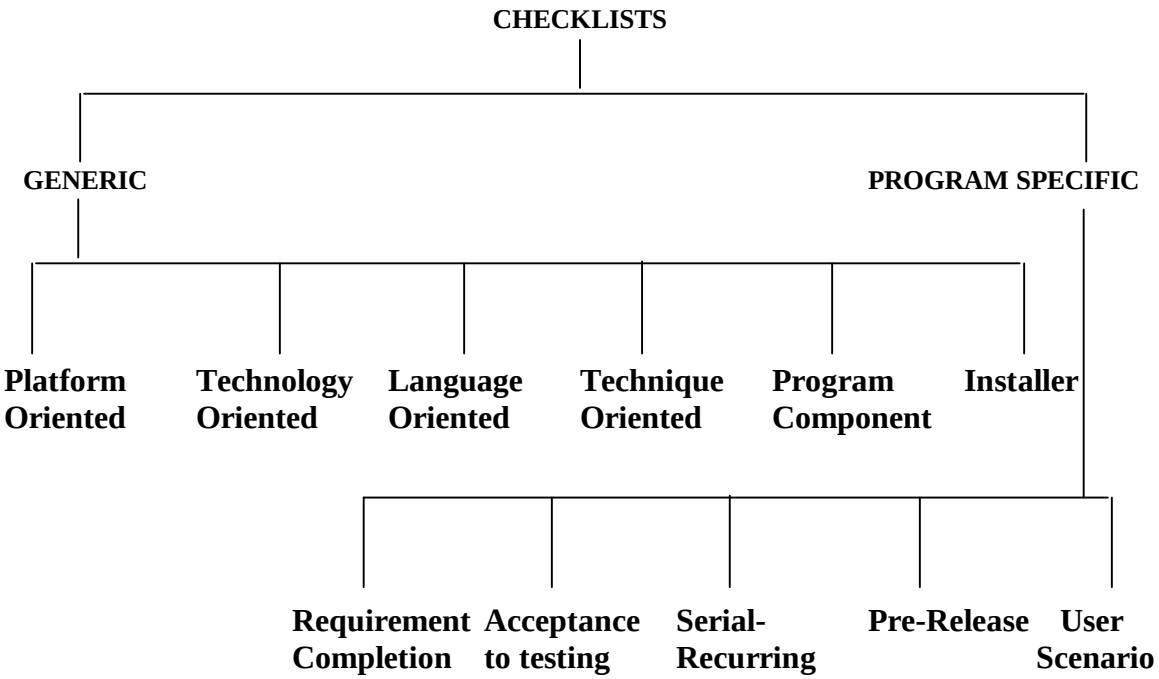
Easy to Update

If a new entry is to be made to the checklist, then testers know exactly which checklist they should be updating. This reduces time for updating and ensures that checks are correctly entered

The Basic Classification …

At broad level, all checklists can be divided into two categories – Generic and Specific. Generic checklists are reusable across projects of similar kinds whereas Program Specific checklists are suitable for a single project. Generic checklists are predefined. Specific checklists grow during execution of projects.

Fig 1.1 describes different types of checklists, which can form part of the two basic categories.



Generic Checklists

Generic checklists can be further classified into the following types. For example, a checklist on Mac-Windows port testing would make an entry in 'Technique-oriented Checklist'. Hence this could be applied to any project that falls into the category of Mac-Win porting.

1. *Platform-oriented checklists* contain checks pertaining to specific platform/OS like Windows, Macintosh, and PALM etc. For example on PALM OS, testing on various ROM is a critical test, which is not applicable for Win/Mac applications.

2. *Technology-oriented checklists* will specify tests related to a particular technology and program architecture. For example, Web, Client-Server, stand-alone desktop application, Wireless, Handheld etc.

3. *Language-oriented checklists* will be based on checks specific to programming language. Technical approach related to database access, User Interface control handling etc., differ from language to language.

4. *Technique-oriented checklists* would describe typical tests for Porting, Migration, Upgrades, etc.

5. *Program-Component checklists* would describe tests for various components of software application, e.g. UI testing, Database testing, Hardware interface testing (Printers, Scanners etc.), Software dependency testing (Mail, Text editor or Browsers invoked from inside application)

6. *Installation checklist* contains typical points related to installation/un-installation of software. There would be checks regarding Registry entry, Installation folder, Program shortcuts etc.

Program Specific Checklists

As already specified, these checklists help you test a single project and may not be usable across projects. Such checklists can be prepared and used at different phases in software lifecycle. Lets have a look at the variety of checklists that can be considered for each program at different phases in its lifecycle.

1. *Requirement Completion checklist* is prepared at Requirement phase i.e. after requirements are frozen and signed off by clients, QA and Engineering groups. This checklist is similar to a Feature List where all major functionalities are understood from Requirement document and translated into Features, which are desired in the application. This checklist helps tester keep track of feature coverage in the project at

any stage. This checklist can be used until Beta when all functionalities are incorporated to the application. It also helps determine closure criteria.

2. *Acceptance-to-Testing Checklist* describes checks that need to be tested on each build delivered to QA, before it is accepted for testing. Checks like correct installation, first-run reliability, build stability, targeted feature completion etc are some probable examples. Based on this checklist, testers can reject or accept a build for testing it further. This checklist is defined between Design and Coding stage for each module and used at the testing phase of the same module.

3. *Serial-Recurring Checklist* defines all the tests, which are periodic and expected to be tested in certain intervals, if not in all builds. It basically contains regression tests of older builds, which have a probability of being buggy because of new feature implementation. This checklist is incremental in nature and grows with addition of new modules. Serial-Recurring checklist is defined in the development phase and grows with every module.

4. *Pre-Release Checklist* contains typical tests regarding clean-ups and stability issues before a release is made to the client. Checks like Debug Message removal, Database Cleanups; Virus checks etc are typical points in this checklist. This serves more as a reminder for the routine procedures that needs to be done before every release. This checklist can be prepared in the initial phases of testing and used before every client release.

5. *User Scenario Checklists* are used at final stages when user-centric testing is done. This contains tests to support use-cases that testers may have already prepared for testing user scenarios. Typical checks for this checklist will be Duration of application usage, simulate user actions, actual flow of usage, restrictive conditions like low internet speed, lower-end hardware configuration etc. This checklist should be prepared in middle phases of testing, when testers get good knowledge of the application.

6. *Staggered Checklists* are distributed tests, which are repeated at regular intervals. These could be Daily, Weekly or even Monthly checks. Such checklists are useful in order to avoid monotony in checking repeated tests, still ensuring regression coverage. Daily checklists ideally have lesser entries compared to Monthly ones. Important checks should be more frequent and find its place in daily/weekly checklist.

Critical Decisions

When a test organization decides to adopt checklist-driven testing, there are few critical points to be considered for its success.

It's important to pick up checklists suitable for the specific project well in advance. Before testers do the actual testing, they should know which checklists they can use for the particular phase in testing. Checklists that are dynamically prepared and are used during testing also need to be determined before testing cycle commences. The whole idea is to plan and systematize the implementation of checklists in advance.

It is important to realize that checklist driven testing cannot survive by itself, nor can it satisfy the total testing need in any application. Checklist should be used along with other testing approaches in right combination. Checklist driven testing should not drive the testing to that point where testers do not get time for other important tests.

Conclusion

Checklists provide the test team with a gamut of testing checks that could be used in a variety of applications. But before an organization decides upon using the checklist driven testing methodology, it is of utmost importance to take out time to systematize the different checklists present. A little care and smart blend of the checklists with the other testing methods can ensure a smooth deployment of this strategy.

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

---