Software Testing Best Practices

Ram Chillarege
Center for Software Engineering
IBM Research

Abstract:

This report lists 28 best practices that contribute to improved software testing. They are not necessarily related to software test tools. Some may have associated tools but they are fundamentally practice. The collections represent practices that several experienced software organizations have gained from and and recognize as key.

1. Introduction

Every time we conclude a study or task force on the subject of software development process I have found one recommendation that comes out loud and clear. "We need to adopt the best practices in the industry." While it appears as an obvious conclusion, the most glaring lack of it's presence continues to astound the study team. So clear is its presence that it distinguishes the winners from the also-ran like no other factor.

The search for best practices is constant. Some are known and well recognized, others debated, and several hidden. Sometimes a practices that is obvious to the observer may be transparent to the practitioner who chants "that's just the way we do things." At other times what's known in one community is never heard of in another.

The list in this article is focused on Software Testing. While every attempt is made to focus it to testing, we know, that testing does not stand alone. It is intimately dependent on the development activity and therefore draws heavily on the development practices. But finally, testing is a separate process activity -- the final arbiter of validity before the user assesses its merit.

The collection of practices have come frohm many sources -- at this point indelibly blended with its long history. Some of them were identified merely through a recognition of what is in the literatures; others through focus groups where practitioners identified what they valued. The list has been sifted and shared with increasing number of practitioners to gain their insight. And finally they were culled down to a reasonable number.

A long list is hard to conceptualize, less translate to implementation. To be actionable, we need to think in terms of steps -- a few at a time, and avenues to tailor the choice to our own independent needs. I like to think of them as Basic, Foundational, and Incremental.

The Basics are exactly that. They are the training wheels you need to get started and when you take them off, it is evident that you know how to ride. But remember, that you take them off does not mean you forget how to ride. This is an important difference which all too often is forgotten in software. "Yeah, we used to write functional specification but we don't do that anymore" means you forget to ride, not that you didn't need to do that step anymore. The Basic practices have been around for a long time. Their value contribution is widely recognized and documented in our software engineering literature. Their applicability is broad, regardless of product or process.

The Foundational practices are the rock in the soil that protects your efforts against harshness of nature, be it a redesign of your architecture or enhancements to sustain unforeseen growth. They need to be put down thoughtfully and will make the difference in the long haul, whether you build a ranch or a skyscraper. Their value add is significant and established by a few leaders in the industry. Unlike the Basics, they are probably not as well known and therefore need implementation help. While there may be no textbooks on them yet, there is plenty of documentation to dig up.

The Incremental practices provide specific advantages in special conditions. While they may not provide broad gains across the board of testing, they are more specialized. These are the right angle drills -- when you need it, there's nothing else that can get between narrow studs and drill a hole perfectly square. At the same time, if there was just one drill you were going to buy, it may not be your first choice. Not all practices are widely known or greatly documented. But they all possess the strength that are powerful when judiciously applied.

The next sections describe each of the practices and are grouped under Basics, Foundational, and Incremental.

2. The Basic Practices

- Functional Specifications
- Reviews and Inspection
- Formal entry and exit criteria
- Functional test - variations
- Multi-platform testing
- Internal Betas
- Automated test execution
- Beta programs
- 'Nightly' Builds

Functional Specifications

Functional specifications are a key part of many development processes and came into vogue with the development of the waterfall process. While it is a development

process aspect, it is critically necessary for software functional test. A functional specification often describes the external view of an object or a procedure indicating the options by which a service could be invoked. The testers use this to write down test cases from a black box testing perspective.

The advantage of having a functional specification is that the test generation activity could happen in parallel with the development of the code. This is ideal from several dimensions. Firstly, it gains parallelism in execution, removing a serious serialization bottleneck in the development process. By the time the software code is ready, the test cases are also ready to be run against the code. Secondly, it forces a degree of clarity from the perspective of a designer and an architect, so essential for the overall efficiencies of development. Thirdly, the functional specifications become documentation that can be shared with the customers to gain an additional perspective on what is being developed.

Reviews and Inspection

Software inspection, which was invented by Mike Fagan in the mid 70's at IBM, has grown to be recognized as one of the most efficient methods of debugging code. Today, 20 years later, there are several books written on software inspection, tools have been made available, and consulting organizations teach the practice of software inspection. It is argued that software inspection can easily provide a ten times gain in the process of debugging software. Not much needs to be said about this, since it is a fairly well-known and understood practice.

Formal Entry and Exit Criteria

The notion of a formal entry and exit criteria goes back to the evolution of the waterfall development processes and a model called ETVX, again an IBM invention. The idea is that every process step, be it inspection, functional test, or software design, has a precise entry and precise exit criteria. These are defined by the development process and are watched by management to gate the movement from one stage to another. It is arguable as to how precise any one of the criteria can be, and with the decrease of emphasis development, process entry and exit criteria went out of currency. However, this practice allows much more careful management of the software development process.

Functional Test - Variations

Most functional tests are written as black box tests working off a functional specification. The number of test cases that are generated usually are variations on the input space coupled with visiting the output conditions. A *variation* refers to a specific combination of input conditions to yield a specific output condition. Writing down functional tests involves writing different variations to cover as much of the state space as one deems necessary for a program. The best practice involves understanding how to write variations and gain coverage which is adequate enough to thoroughly test the

function. Given that there is no measure of coverage for functional tests, the practice of writing variations does involve an element of art. The practice has been in use in many locations within IBM and we need to consolidate our knowledge to teach new function testers the art and practice.

Multi-platform Testing

Many products today are designed to run on different platforms which creates the additional burden to both design and test the product. When code is ported from one platform to another, modifications are sometimes done for performance purposes. The net result is that testing on multiple platforms has become a necessity for most products. Therefore techniques to do this better, both in development and testing, are essential. This best practice should address all aspects of multi-platform development and testing.

Internal Betas

The idea of a Beta is to release a product to a limited number of customers and get feedback to fix problems before a larger shipment. For larger companies, such as IBM, Microsoft and Oracle, many of their products are used internally, thus forming a good beta audience. Techniques to best conduct such an internal Beta test are essential for us to obtain good coverage and efficiently use internal resources. This best practice has everything to do with Beta programs though on a smaller scale to best leverage it and reduce cost and expense of an external Beta.

Automated Test Execution

The goal of automated test execution is that we minimize the amount of manual work involved in test execution and gain higher coverage with a larger number of test cases. The automated test execution has a significant impact on both the tools sets for test execution and also the way tests are designed. Integral to automated test environments is the test oracle that verifies current operation and logs failure with diagnosis information. This is a best practice fairly well understood in some segments of software testing and not in others. The best practice, therefore, needs to leverage what is known and then develop methods for areas where automation is not yet fully exploited.

Beta Programs

(see internal betas)

'Nightly' Builds

The concept of a nightly build has been in vogue for a long time. While every build is not necessarily done every day, the concept captures frequent builds from changes that are being promoted into the change control system. The advantage is firstly, that if a major regression occurs because of errors recently generated, they are captured quickly. Secondly, regression tests can be run in the background. Thirdly, the newer releases of software are available to developers and testers sooner.

3. Foundational

- User Scenarios
- Usability Testing
- In-process ODC feedback loops
- Multi-release ODC/Butterfly profiles
- Requirements for test planning
- Automated test generation

User Scenarios

As we integrate multiple software products and create end user applications that invoke one or a multiplicity of products, the task of testing the end user features gets complicated. One of the viable methods of testing is to develop user scenarios that exercise the functionality of the applications. We broadly call these User Scenarios. The advantage of the user scenario is that it tests the product in the ways that most likely reflect customer usage, imitating what Software Reliability Engineering has for long advocated under the concept of Operational Profile. A further advantage of using user scenarios is that one reduces the complexity of writing test cases by moving to testing scenarios than features of an application. However, the methodology of developing user scenarios and using enough of them to get adequate coverage at a functional level continues to be a difficult task. This best practice should capture methods of recording user scenarios and developing test cases based on them. In addition it could discuss potential diagnosis methods when specific failure scenarios occurs.

Usability Testing

For a large number of products, it is believed that the usability becomes the final arbiter of quality. This is true for a large number of desktop applications that gained market share through providing a good user experience. Usability testing needs to not only assess how usable a product is but also provide feedback on methods to improve the user experience and thereby gain a positive quality image. The best practice for usability testing should also have knowledge about advances in the area of Human Computer Interface

.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.