Software quality challenges and practice recommendations

Aya R. Elgebeely June 25, 2013

Aya Elgebeely discusses the importance of four key quality assurance practices in software development.

Introduction

Software development and engineering are considered fairly young professions; however, they are widely practiced and growing faster than ever before. The software industry in general is now considered one of the main pillars of economic growth in many countries. Software companies frequently face many difficult challenges to deliver high-quality software, and they strive to achieve customer satisfaction.

Software quality as a necessity

As software became an indispensable part of our daily lives, demand for software increased significantly. Accordingly, high software quality is now perceived as a "must have" rather than "should have." It's essential to involve quality assurance teams in the project planning and execution from the very beginning. Yet companies that consider software quality as merely a task that is done by testers at the end of the development lifecycle still exist in our world.

It's worth noting that the software market is full of alternatives, and there is plenty of free, open source software out there. Besides that, customers and end users are becoming more aware of the quality of the software that they buy. Applications or enterprise systems that show poor performance or low-quality user experience will be eliminated, and other products will take their place easily. Now it's the mission of software companies to take care of the quality of their products as never before.

Software quality considered from beginning to end

The problem with the approach of abbreviating software QA into testing phase after all development tasks are done is that it costs the team a lot and puts the whole project at high risk. During the testing phase, the developers do their best to assure that their code has minimal defects. Then the tester works hard to uncover every possible defect in the software, while the managers and clients hope that they have software ready to release to market.

The question is: Why do many software companies insist to pushing the development team to meet strict deadlines and finish as many features as they can, with no regard to how poor the code is, neglecting the large amount of defects injected inside the code, making architectural mistakes, and ignoring documentations?

Rushing through development might save the team's time at a given moment, but it ultimately takes them more time to do it over if there were major development issues not considered from the beginning. It results in wasting a lot of the team's resources in fixing and re-engineering their code instead of investing those resources in something more useful. Software teams know the whole story by heart, but with nagging clients and strict sales team accompanied by some developers' egos that they write defect-free software, it's really hard to escape dropping QA aside to rally behind just finishing the code.

Software engineering standards and their use

It's worth mentioning that companies need not necessarily follow one of the software development standards nor have a strict process in place. There are various standards for the typical software development life cycle (SLDC), such as IEEE, ISO - 12207, or CMMI. The aim of these standards is to make sure that the end product complies with market requirements and attains end user satisfaction.

In fact, many software applications, mobile apps, and even full enterprise systems are sold to various customers every day that might not have been developed using any standard. However, people buy them anyway. Ignoring standards is not directly proportional to the poor software quality and lesser demand for the end product (as long as it's not life-critical software, such as medical software that requires FDA approval inside the US and should be compliant with one of the standards). The problem is not in following standards. What really matters is ignoring or diminishing the importance of the quality of the software.

The point of this article is not about SDLC standards nor having a superb development and testing process. First, it is important to realize that quality is a crucial component of any software. Companies do not necessarily need to have a highly professional QA team and practices, but at least the culture itself must be adopted and relevant practices should be in place.

Software quality practices through the software development cycle

This section presents practices that would positively affect software quality without creating too much load or headaches for the development team. They are worth considering in development and testing practices.

Requirements review

Don't you agree that it's really awful to waste your resources in delivering the wrong feature? Reviewing the software requirements before starting each new development phase minimizes defects and fulfills clients' needs. Reviewing requirements before implementation helps in considering potential changes and overcoming misunderstandings that might occur throughout

the project lifespan. The team has to double-check with the customer all business domain details that should be implemented. Requirements review can also be done using prototypes and domain models. When the development team does this small assignment before starting the actual implementation, they get excellent kickoff of their project or development iteration. By making sure that all stakeholders reach a consensus and every team player is on the same ground before rushing into implementation, the client and management are sure that the developers will deliver the right thing at the end of the development cycle.

Code review and walkthrough

As simple as it sounds, code review is one of the most effective practices in software development. It has direct impact on reducing the number of defects (to catch the bug at the window before it enters the house) and enhancing the quality of the code and software design. This diminishes the need to do major code refactoring and clean-up in future releases.

The team can agree on simple coding and design guidelines according to the project requirements and implementation details. These guidelines should be shared among the team members, and whenever a new feature is developed, one or more team members (other than the author) should review the new code and search for any coding or design mistakes.

This practice helps the team in many ways, including enhancing code quality and design, minimizing defects, and preventing them. Also, it allows the whole team to gain insight into each other's work, eases handover, and increases team awareness about different software components and functions. The team collaborates to verify and validate the quality of the code and how the design was implemented. They get direct feedback from their peers. There is a double benefit here: The code quality increases, and the team awareness and project ownership does, too.

Session-based testing

Session-based testing, a methodology developed by James Bach, means dividing the testing load into sessions, where each session has a mission (a clearly stated outcome desired from the testing session). Each session has a defined timeframe (from 20 to 40 minutes), and the tester should be uninterrupted while the testing session is conducted.

It's like putting the tester inside a testing bubble for a while and letting the tester focus on finding defects for a specific software feature or function. During the session, the testing is guided by a set of test cases, and the tester can do exploratory testing, as well. Thus, session-based testing is a mixture of a formal testing method and testing innovation, because it gives the tester room for exploration and intuition that can allow time and leeway to find unusual defects or mess around with the software to get more from it.

During the session, the tester should document the behavior of the software, take snapshots, and write down the behavior of the software under specific input and setup. After the session ends, the session transcript is discussed with the team leader or technical manager. From their discussion, they find out what is considered a normal behavior and what is not, and then defect reports are created, based on this discussion.

Figure 1 describes session-based testing methodology briefly. For any new change in the software, different test sessions are planned, each with a specified goal and mission. During the testing session, the tester uses test cases or does exploratory testing, or both. When the testing session ends, the defects found during the session are reported.

Figure 1. Session-based testing workflow

Risk-based testing

Development teams usually have frequent releases for the same software, because a lot of changes — either major or minor — happen during the development process. An important QA practice is to thoroughly test the software after each major release. On the other hand, it is time-consuming and difficult to run a full regression test suite for the whole software for each release. However, it is unsafe to test only the changed features or cut down the test case suite awkwardly. A piece of code might solve a defect but break something else in the code.

The risk-based testing approach stands in the middle. Its basic idea is to sort the software features and failure modes in descending order, from the most important or riskiest to the nice-to-have features and simple risks (one of the tools to do this is FMEA: failure modes and effects analysis). When the tester has such list at hand when testing a new release under a tight schedule, the tester focuses the efforts to make sure that the newly introduced changes didn't break anything else. Ten it's easy to make sure that the changes didn't break any of the most important features in the software and that none of the most severe risks occurred.

Summary

Every company aims to fly high in the competitive IT market, and it takes effort to make good software that people like. Unfortunately, sometimes stress from clients or impatient management

might lead teams to bypass software quality practices, and they could end up delivering a product that is below expectations. The poor quality of the software is noticed only when it fails.

The practices and methodologies discussed in this article span the development lifecycle. They cover the requirement analysis, design and development, and testing phases. And they show that defect prevention is much easier than putting out fires at the end of the project, when the technical debt and cost of changes are high.

In this article, the software quality role and importance are highlighted to assure that ignoring software quality could significantly impair companies from achieving their business objectives. Also, the article introduces the reader to some of the more lightweight and efficient practices that save the team time, money, and effort while boosting the product's quality.

Related topic

- Download a free trial version of Rational software.