STAR Tester

EuroSTAR 2007
3 - 6 December
Stockholm

Send to a Friend »

Cover Page »

In This Issue

Sponsors

Useful Links

Subscribe Now
EuroSTAR
Software Test Training
EuroSTAR Expo
Quick Guide to EuroSTAR
Our Blog

Testing In A Quasi-Agile¹ Software Development Environment

by Timothy Korson, QualSys Solutions, USA

What! No Update or Delete?

Figure 1 illustrates the philosophy taught in many software engineering courses. It reflects a widespread "head in the sand," mistaken belief about the way software systems can be developed.

Figure 1.

According to this commonly held view, requirements are input to the software development process and are ideally fully developed, verified, and frozen prior to the start of design or coding. In this world, the software development team's responsibility is to transform an externally provided set of written requirements into an executable software system. The problems with this idealized scenario are at least threefold:

First, most requirements are based on a cost benefit trade-off analysis. But until some design work is done, costs may not be knowable. In general, meaningful cost benefit analysis cannot be done without the involvement of the technical software engineering staff.

Second, most stakeholders are unaware of the capabilities of modern software systems, so they end up specifying sub-optimal systems based on limited knowledge.

Third, and perhaps most importantly, humans are notoriously incapable of specifying in the abstract. A short example will illustrate my point. Suppose my wife needs a new dress for an important event and sends me to the mall to get one for her. To ensure that I acquire a dress fitting for the occasion and that she will like, she writes a detailed formal requirements document, specifying preferences for style, size, color, fabric, ease of cleaning, wrinkle properties, cultural expectations for the type of event and season of the year, and every other type of functional and quality requirement that could be imagined. What do you think the chances are that the dress I bring home will fulfill her expectations? If you would take on a task like this, you're a braver soul that I am, yet we attempt similar impossibilities in the software world all the time.

The problem is that humans are very poor at specifying or even knowing what they would like until they see something concrete. Most people can't even tell if they like an existing article of clothing on the rack in front of them until they try

Archive

Issue 34
January 13, 2009
Issue 33
December 2, 2008
Issue 32
November 4, 2008

More »

Contact Us

Email
Tel: +353 91 514478
Website

it on. Software is the same way. We need to continually "try it on" if we are to end up with a system that delivers value. This is a major rational behind iterative/incremental software development processes. Effective software development practices must allow users to periodically experience the software and evolve the specifications as the system takes shape.

With these realities in mind, Figure 2 includes requirements definition and evolution within the software development process.

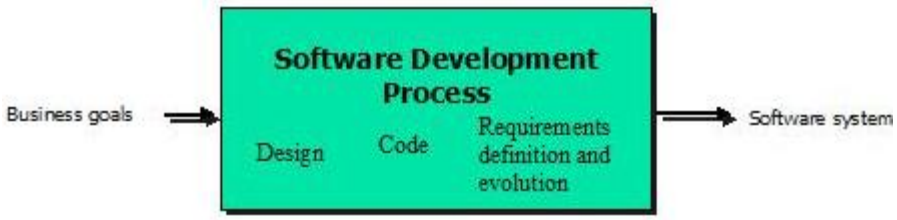

Figure 2.

In this scenario, in addition to the technical tasks of software engineering, the software development team is responsible for collaborating with the stakeholders to facilitate the definition of requirements that meet the business goals of the system. In the agile world we often put it this way: "The goal of software development is to deliver value to the client."

The difference in the two approaches is also apparent in the way we define quality. In the world of Figure 1, quality is usually defined as meeting the requirements. In the world of Figure 2, quality is defined as that which provides value to the user. I'll illustrate the difference with an example from a recent experience.

An organization that hosts conferences where I lecture, wanted their website to have the ability to give discounts in various situations, such as when three or more employees from the same company sign up for a given event. To allow for this, the requirement of being able to create promotion codes was added to the specifications for the website. In due time this feature was implemented and used by the owners of the website. However, when the web administrator went to try and change the percent discount for a promo code, he found that he couldn't figure out how to do it. Furthermore he found he couldn't even find a way to delete any of the already added promo codes. When the website development organization was called and asked about how to do modification and deletions to the promo codes, the answer was, "you can't. You didn't ask for updates or deletions, your request was for the ability to add promotion codes and that is what we gave you."

According to the traditional views of figure one, the website company acted properly and delivered a quality system. According to the agile viewpoint as illustrated in figure two, the website company acted irresponsibly and delivered a poor system. What do you think?

Where does this leave you as a tester? How do you view quality? What is your role?

The role of a tester in Figure 1 is well defined. It is to verify that the system correctly implements all of the written requirements. This will include both functional and quality requirements, but is limited to what is specified in the requirements document. The job of a tester in this environment might be tedious or exhausting, but it is well defined.

The role of an agile tester focuses on ensuring that the system being built delivers value to the stakeholders. It acknowledges that waterfall requirements will always be incomplete, inconsistent, and many times simply wrong. It means the tester must be intimately familiar with the business goals of the system and understand what provides value to each of the stakeholders. To achieve this level of insight, the tester must be thoroughly involved in all project activities.

When an agile project is done correctly, it includes processes such as test first development and pair programming. These practices help rid the code of basic bugs and means that the code given to the tester usually does what the developer intended it to do. This frees the tester to focus on validating that the system delivers value to the client.

In the website promo codes example above, the development team should have realized that the client needed update and delete as well as create, but the reality is that programmers have a focused mindset. As soon as a requirement is even partially articulated, the programmer is brainstorming on the best way to implement it. The client should have been more precise in the requirement, and clearly specified update and delete as well as create, but the reality is that clients know their own needs so well that they will always make assumptions about what is so obvious that it would be redundant to spell it out.

Testers are a special bread. We sit in the middle and are in the best place to bridge the gap. We have to deal with the code and define detailed test cases, so we tend to be able to anticipate how programmers think. But we also deal with the system as the client's advocate, so we know how they think. Because we are able to understand how both sides think we have a unique opportunity to add value.

The reality of software development is that clients will always neglect to explicitly specify what is obvious to them. Programmers will always focus on algorithms and implementation. It is up to us as testers to go beyond written requirements and ensure that the system really delivers value to the stakeholders. Fortunately the tools of our trade such as equivalence classes, decision tables, state diagrams, scenarios, and negative testing help up to systematically consider variables and situations that might fall through the cracks without us.

[1] A quasi-agile environment is one where software developers are trying to implement agile values and practices within a traditionally structured organization that has policies and procedures derived from waterfall concepts.

Biography
Timothy Korson, QualSys Solutions, USA
Timothy Korson has had over two decades of substantial experience working on a large variety of systems developed using modern software engineering techniques. This experience includes distributed, real time, embedded systems as well as business information systems in an n-tier, client-server environment. Dr. Korson's typical involvement on a project is as a senior management consultant with additional technical responsibilities to ensure high quality, robust test and quality assurance processes and practices.

Click here for full tutorial description

Printer Friendly Version »

Published by EuroSTAR