Software Quality Beyond Testing In-house Code

Klaus Haller & Rudolf Grötz

Abstract

Crucial software fails and management needs someone to blame? Blame the testers! They should find bugs before they get into production! Yes, even testers make mistakes. They might even sign-off software they are not convinced of. More often, however, issues are not caused by bugs in the tested code, but by other factors. This article discusses three of these factors. First, there are configuration parameters. They impact test coverage and test processes. Second, relying on suppliers implies specific quality risks. Third, the business aims themselves can cause issues. The aim of this paper is to discuss these three points and to provide a solution by enhancing standard software change processes.

Configuration Parameters: When software suddenly turns mad!

Configuration parameters allow the adaption of software behaviour quickly, if business needs change. Also, they ensure repeatable installations. Repeatable means that when the installation is complete, it is in a carefully defined state; it can be reconstructed for future test cycles. Thus, all installation parameters such as paths, Java Virtual Machine settings, timeout periods *etc*. must be put into an installation parameters file. Based on this file, a batch job performs the actual installation. This is one core idea of DevOps [1]. When test and production systems are set up as similar as possible, this reduces “production only bugs.” Such bugs appear in production only and do not appear in testing due to a different set-up. They are the fear of IT departments.

However, besides technical parameters, there are Application parameters. They impact the business logic. In a core-banking system, they define *e.g.* the limit for loans for which two credit officers have to approve the loan. Other parameters provide the files of the bank logo used for account statements. Parameters provide more flexibility since changing them is easier than changing code.

On the other side, such parameters have drawbacks. Test coverage can drop and they allow for bypassing the software change processes. Various software deployment tools install software in production only, if it is packaged and signed-off by testing. Neither developers (and certainly not users) can change the software behaviour without a sign-off from testing. However, this changes for GUI parameters. Power users might be able to change GUI parameters and, thereby, bypass the software change process (Figure 1).

An organizational solution is needed: First, restrict who has access to GUI parameters. Second, communicate that no change is allowed without testing. Third, make clear that sanctions for not following the rules are widely understood.

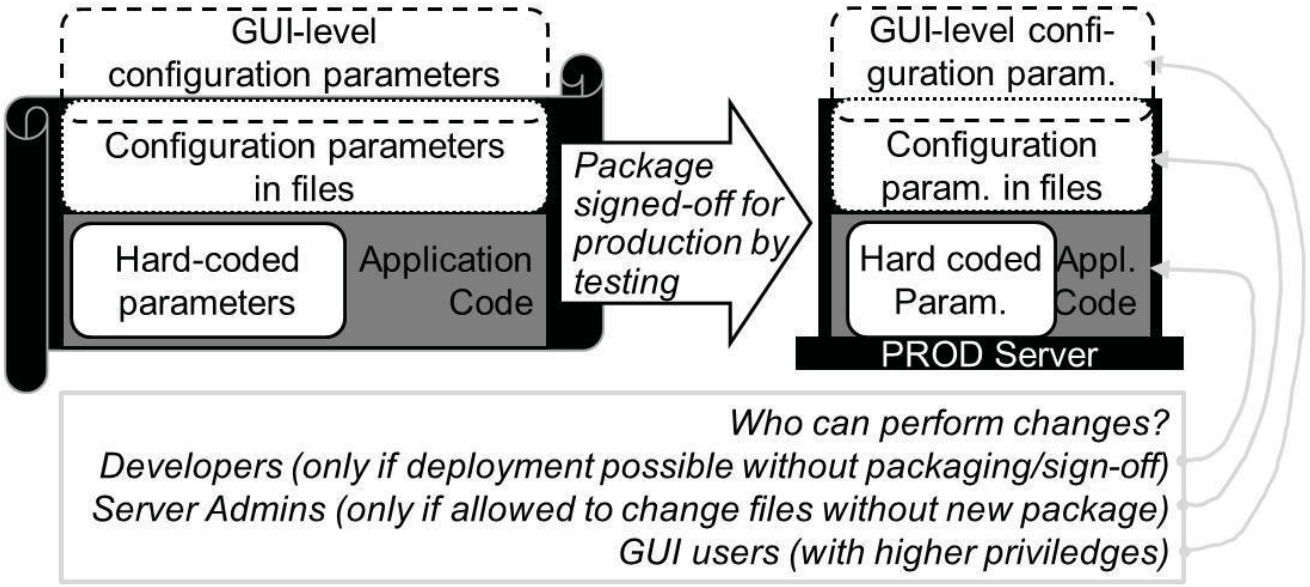

Figure 1: How configuration parameters undermine test and change processes

The second drawback is a drop in test coverage. The number of configuration options might explode due to the parameters. No test budget will grow at the same pace. We assume a system with five parameters: JVM memory settings, timeouts, maximum number of users, disk size and application server version. Each parameter can have one value out of four. The result is 4*4*4*4*4 = 1024 configuration options. No IT department will pay for testing all options if the software is installed in three branches in Zürich, London, and Singapore only.

Thus, there is a risk that when changing parameters in production, the new configuration might not have been tested. The application usage can move out of the test scope (see Figure 2). It is not clear whether the software might crash or produce wrong results. To prevent this, changing parameters must trigger testing, even if there is no new package (Figure 3, Checkpoint A).

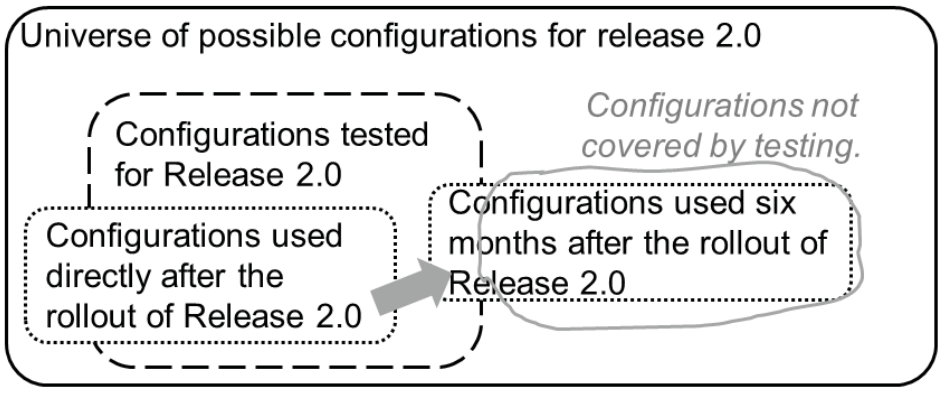

Figure 2: Test Coverage and Application Usage over time.

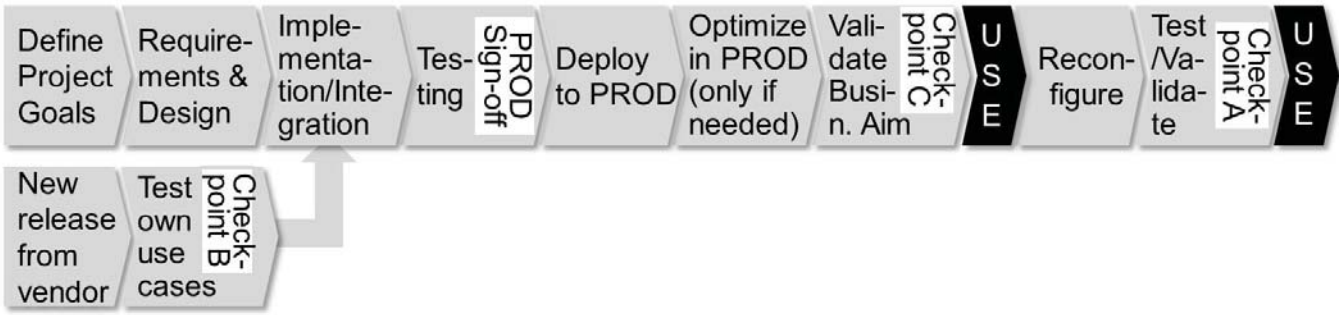

Figure 3: Unified software change process considering technical and business-focused quality assurance

Software Supply Channels – Stable as a House of Cards?

This section looks on the quality impact of 3rd party software components. Our example is a contract management solution of an insurance company. It enables insurance agents to print out contracts, which clients sign. It can scan contracts and store them in an archiving system. The solution incorporates three 3rd party software components: a reporting engine for rendering a PDF with the contract for printing it; a scanning solution with OCR; and a document archive (Figure 4).

All vendors have one dilemma in common. On the one hand, they need economies of scale. The software must meet the needs of many (potential) customers. On the other hand, software vendors make an implicit promise: the software works; it is (nearly) bug-free; you can start using it tomorrow. Obviously, the more configuration options software has, the less likely is that all options are tested in-depth and work as expected.

The dilemma of software vendors has implications for IT departments. First, the latter have to accept this reality. Vendors test a new release before rolling out software to their customers. Their test scope, however, is not guaranteed to match the exact usage scope of all customers. Second, IT departments must manage this quality risk. They could hope that there are no bugs or that those that are present are found in system integration testing. This is obviously late and risky. A better approach is to model test cases based on their own usage of the 3rd party software. The IT department tests based on them when the vendor rolls out a new release. This is a new quality gate (Figure 3, checkpoint B). Only if the new release works with the rest of the solution, is it incorporated in the customer's IT landscape.

In the case of niche products, the most sustainable solution is to try to hand over the test cases to the vendor. Then, the vendor can add them to their regression test set.

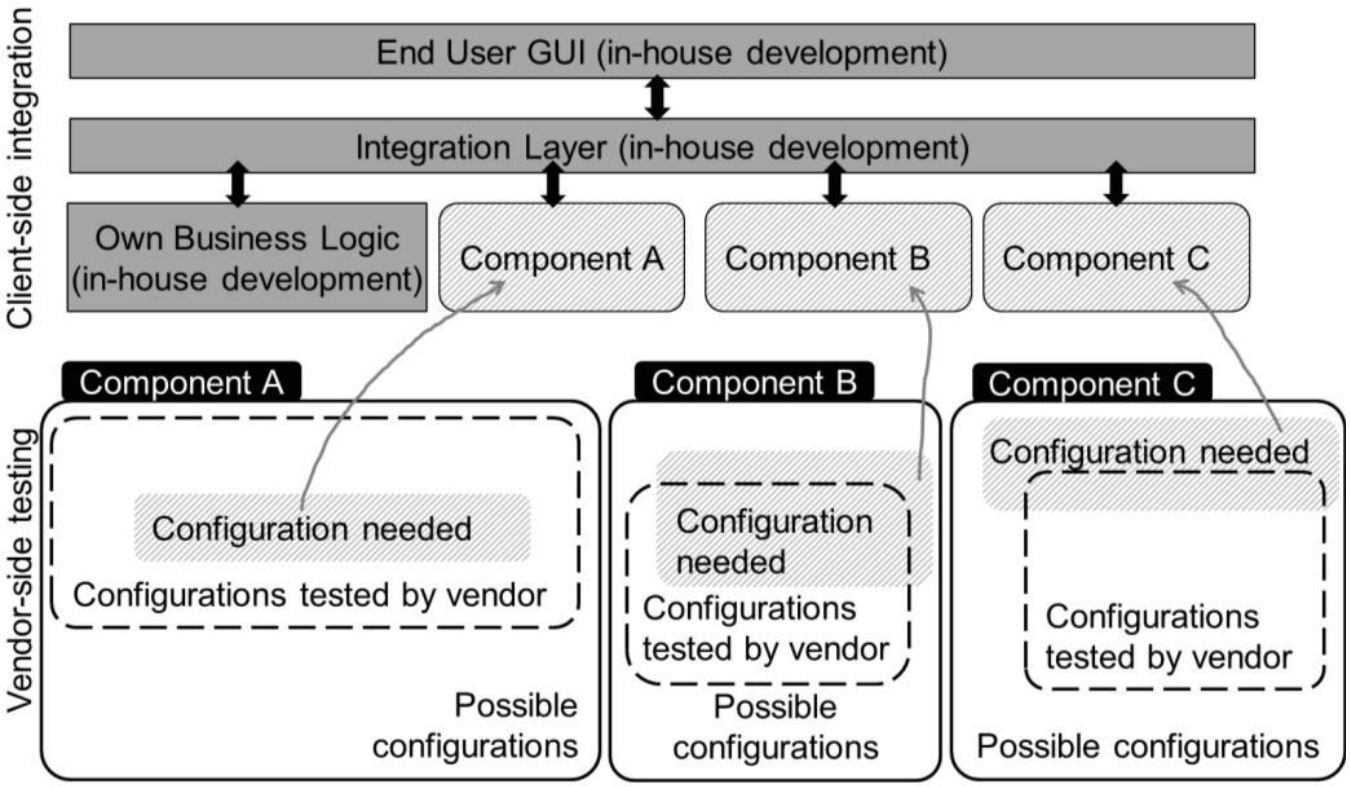

Figure 4: Quality risks in software supply chains

Business Focused Quality – or – Testing is neither Optimization nor Validation of Business Aims

The reason to start an IT project can be anything from a purely technical to a highly business-related aim. An example for a technical project is upgrading all Linux servers to a common patch level. The project succeeds if the technical goal is reached. Projects with a strong business focus differ. We use an investment fund as an example. The fund uses an automated trading system, which decides on a day-to-day basis when to buy and sell which stocks. Now a trader has an idea: If we introduce a new trading rule “sell stocks which gained 10% or more in a week”, the fund profit should rise by 1% per year.

Three dimensions describe the success of the project:

(1) Technical correctness: Is the business rule implemented as specified? Are stocks sold if they gain 10% or more in a week, but not if they raise only 5% or drop by 15%?

(2) Achievement of business aim: Does the new rule increase profits by 1%?

(3) Optimization question: Is “10% gain within a week” the best configuration? Could the profit be increased by changing the rule to “sell stock if it gained 8% within three days”?

Testers sign-off the technical correctness of the software after testing (1). They do not and cannot check whether and how efficient software helps achieving business aims (2 and 3). For the latter, often the software has to be in production for days or weeks to see the effects. This requires rethinking root-causes for rolling back to an old release or deploying emergency fixes to production.

IT problems (buggy software, which get into production, aka a testing disaster) are only one root-cause for emergency fixes. Wrong assumptions by the business are a second option (*e.g.* the trading rule was not a good idea). The software change process has to reflect them as well. Besides a sign off from testers, a sign-off against business aims and

optimization goals is needed. This requires adding a new checkpoint to the software change process *after* the deployment to production (Figure 4, checkpoint C).

Conclusion

Software quality is more than testing internally developed code. Configuration parameters, 3rd party software components and business (optimization) aims pose new challenges for software testing and change. To overcome this, this article elaborated how to enhance software testing and change management processes to uniformly assure technical and business-focused software quality.

[1] M. Loukides: What is DevOps? http://radar.oreilly.com/2012/06/what-is-devops.html, last retrieved July 26th, 2015

Klaus Haller *is an IT consultant with Swisscom Enterprise Customers in Zurich. Since 2005, he has worked mainly in the Swiss banking sector. His areas of expertise are testing and test centre organization, test data management, compliance testing and IT risk. He publishes frequently in magazines and speaks on conferences. More about him on his webpage http://www.klaushaller.net*

Rudolf Grötz *is an ISTQB Certified Full Advance Tester. He heads the QA Division of Jumio Inc. in Vienna. Since he got in contact with agile ideas in 2008, he is convinced that "agile" is like a poison. It works in the right doses, but too much is deadly. Thus, he continuously aims to find the right does to make requirements engineering and test automation a success. He publishes frequently in magazines and speaks on conferences. More about him on XING: https://www.xing.com/profile/Rudolf_Groetz*

The opinions expressed in this article are the authors' own and do not necessarily represent the views of the companies they are working for.

Write an article

We are always on the lookout for new content, so if you have a testing story you would like to share, a test technique you would like to evangelise or testing research you would like to publish, then The Tester is the place to do it. Simply email the Editor on phill.isles@bcs.org

---