QUASAR #170

INFORMING AND ADVANCING OUR MEMBERS

www.therqa.com

FEBRUARY 2025 | ISSN 1463-1768

SOFTWARE TESTING: MEASURING VENDOR SOFTWARE QUALITY – PART ONE

GCP INSPECTION LANDSCAPE IN CHINA

# BRAINSTORMING – A QUALITY TOOL

30 PAGES OF ARTICLES

MASTERING THE MAZE: CRAFTING A QMS FOR IVD COMPLIANCE

FROM DESIGN TO DATA: ACHIEVING QUALITY IN CLINICAL TRIALS

NAVIGATING THE CHALLENGES: IMPLEMENTING PHARMACOVIGILANCE AUDITS ACROSS DIVERSE REGULATORY ENVIRONMENTS



RQA

Barry McManus

Hugh ONeill

# SOFTWARE TESTING: MEASURING VENDOR SOFTWARE QUALITY – PART ONE

This topic grew out of a discussion about testing techniques versus managing the scope of test documentation. This is the start of a series of articles on computerised system testing. Given that the bulk of testing techniques typically reside within the vendor's domain, the authors aim to explore some aspects of this vast topic area in order to better prepare the QA function when assessing a vendor's test practices and associated software product quality.

## WHAT IS TESTING?

'Software testing consists of the dynamic validation that a System Under Test (SUT) provides expected behaviours on a finite set of test cases suitably selected from the usually infinite execution domain.'[1]

- The SUT is the test object (application, middleware, database, hardware) that comprises some or all of the computerised system
- The term 'finite' suggests given the constraint of resources and schedule, that testing targets a subset of all possible test scenarios determined by various criteria.

In the GxP arena, (acceptance) testing is performed to ensure that the intended 'behaviours' are provided. Quite often this is accompanied by volumes of 'documentation'.

There is a constraint in that the need to deliver 'copious' amounts of documentation, as objective evidence, is consuming the finite resource capacity to test. The CSA guidance[2] hints that this is a problem when it talks about using risk-assessment to 'follow the least-burdensome approach' and 'the burden of validation'.[2] However, if the 'scope' of documentation is reduced, are we able to ascertain the quality of the vendor's computerised system? In other words, is there a correlation between documentation quality and computerised system 'product' quality?

To answer this question, we need to examine the scope of testing as a discipline, how it relates to the software lifecycle and how it can be leveraged to assess software manufacturing processes and associated software product quality. This is the vendor's domain as the bulk of testing occurs within the software lifecycle.

## THE VENDOR PROBLEM

Vendor audits* conducted by the authors have revealed an increasing trend among the vendor community in the following areas:

1. The vendor's QMS focus is on the 'what' is done but not on 'how' it is done (the instructions to realise the 'what'). The QMS doesn't oversee the technical activities that build the software product quality.
2. The vendor's internal audit function focus is on documentation quality rather than on the technical software product quality.
3. The vendor's test focus is (only) on proving their requirements are being met by end to end scenarios, accompanied by generating validation type documentation. When queried as to the rationale for this, the reply was the same: 'Auditors demand this'. (Quite often the vendor may also state that such documentation can be bought to reduce the validation burden for their customers).

*18 vendor audits across 2023/24

A focus on documentation may be noble, but when faults/defects arise during validation and production use of the computerised system, the mantra that 'software always contains faults' is not good enough. Especially where the software product quality hasn't improved across multiple releases (which raises the question on the QMS ability to improve 'quality').

We, as auditors, should not just be 'demanding' a level of test documentation quality from vendors, but consider the level of testing effectiveness to measure the software product quality. To do this, we need to start with the purpose of testing.

## THE PURPOSE OF TESTING

Forgoing that testing is a regulatory requirement and there is a need to pass an inspection, why should we have testing?

'Testing is a process of executing a program with the intent of finding errors.'[3] The purpose of testing is four fold:

1. 'To verify that requirements have been met'.
2. 'To uncover an as yet undiscovered error'.
3. 'To reduce the impact of risks that have materialised in production use'.
4. 'To produce relevant, objective information to make informed decisions.'[3]

The first purpose of testing can be rewritten as 'demonstrating that the system does what it must do'[3]. Focus on this positive perspective can detract from the second purpose of testing, which can be rewritten as 'demonstrating that the system does not do what it is not supposed to do'[3]. This additional view is important as it seeks to challenge the SUT in ways that are outside the bounds of normal operational (correct) use. If the vendor does not focus on demonstrating that the system does not do what it is not supposed to do, then there is a risk of faults/defects slipping into production use.

## TEST STAGES

The scope of testing is dependent on the SUT and the resource constraints. However, typically four test stages are applied: unit, integration, system and acceptance. The first three are within the vendor's domain prior to delivery, the fourth by, or on behalf of, the customer at validation UAT. See Table 1.

**TABLE 1. TEST STAGES**

| TEST STAGE | DESCRIPTION | EXAMPLES OF ISSUES TO PREVENT |
|---|---|---|
| Unit | Is used to verify the individual code logic (units) that are combined together to realise a SUT feature. The unit test challenges the correctness of code logic (in isolation of a feature) and is thus too small to test a feature. It is, more often than not, performed by the person who 'writes' the code logic and is typically written as code. | Logic Errors: Incorrect implementation of algorithms or conditions.<br>Data Boundary Issues: Errors occurring at the edges of valid input data ranges. |
| Integration | Concerns the testing of the incremental interactions between software modules that focus on data communication (flow) between modules. External integrations to other applications, utilities or hardware devices can be considered. | Interface Mismatches: Errors in data format, type or sequence between modules.<br>Data Flow Problems: Incorrect or missing data passed between components. |
| System | Builds on integration testing and focuses on the behaviour of the SUT. In addition to the issues found in unit and integration testing, system testing will assess non functional aspects of the SUT, such as security, speed, usability, reliability and error handling. | As for integration, plus, End-to-End Workflow Failures: Errors in multi-step processes, such as incomplete transactions. |
| Acceptance | This is well known to the regulatory community and seeks to confirm that the deployed SUT meets the end user's expectations and requirements. | Requirement Mismatches: Features implemented incorrectly or missing.<br>Usability Issues: Poor user experience or unclear workflows. |

## TEST STRATEGY

Mature software vendors understand that computerised systems are prone to faults/defects and that testing is a finite activity. As a result, these vendors define test strategies to focus on assessing the level of software product quality.

A generic testing strategy may encompass something along the following lines:

1. Different types of software testing is usually performed at different levels throughout the end-to-end development/validation lifecycle.
2. Different testing techniques are used to efficiently target faults/defects and requirements at the most appropriate point in an end-to-end computerised system lifecycle – regardless of the type of lifecycle.
3. Initial unit testing is conducted by the implementer of software code.
4. Testing and debugging are different activities.
5. Testing beyond this is typically conducted by an independent test group.
6. Static testing is an effective test technique to prevent errors from being designed and built into a system. Static (preventative) testing is as important as dynamic (corrective) testing.
7. Early testing seeks the identification of errors (where the cost of correction is cheapest), later stage testing builds on this to confirm SUT correctness.
8. Testing is expensive and must be managed to ensure the optimum use of finite resources.
9. Effective test management measures the quality of the test process and the computerised system 'product' quality. The measurement of quality within a test strategy is vital as management should ascertain the effectiveness of the strategy to achieve its goals (that is, the four purpose statements overleaf) within the applied finite constraints.

If we are to focus less on documentation quality and more on software product quality, then how should we approach this? ICH Q9 defines quality as 'fulfils requirements.'[4] This is a subjective term and will mean different things to different people.

For this series of articles, quality is defined as: 'Software (product) quality is the absence of defects which would either cause the SUT to stop working or to cause it produce incorrect results.'[3]

## WHY USE THIS DEFINITION?

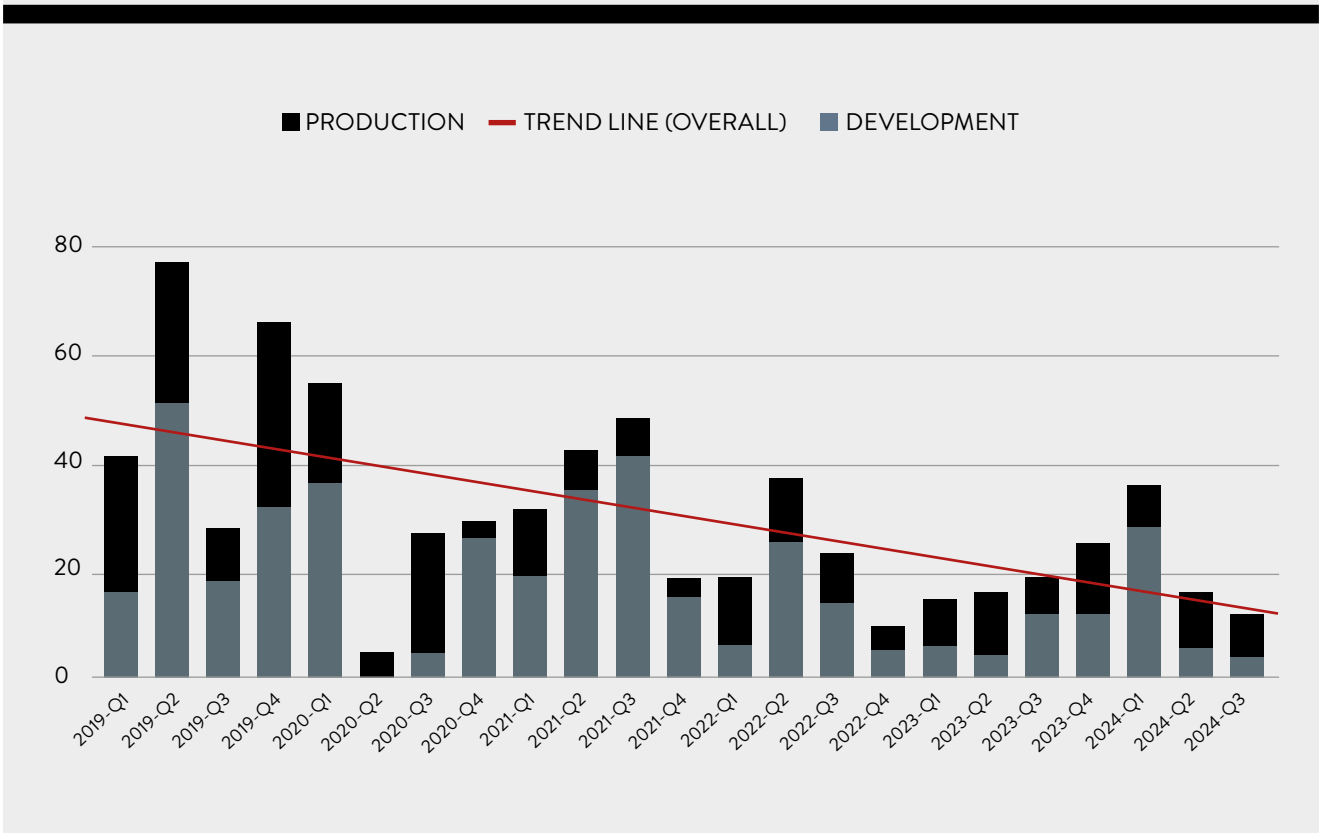
1. Software defects impact the following quality attributes: reliability, maintainability, usability, security, data integrity, fitness for use, conformance to requirements and customer satisfaction.
2. Thus, faults/defects facilitates the measurement of quality. 'When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is… unsatisfactory… you have scarcely advanced the state of science…' (Kelvin)[5].

When we measure the computerised system (and the associated vendor service) then we can measure the schedule, cost and quality of the software product.

For example, consider the following defect bar graph that provides the number of defects raised during software manufacturing and then during production release by end users across quarterly software releases. See Figure 1.

The volume of defects raised by customers in production and by the vendor has been falling across releases. This trend will invariably raise more questions, such as quality per feature or severity, but it is a starting point during vendor discussions when assessing the quality of their software manufacturing line to generate software product quality.

**FIGURE 1. BUGS IDENTIFIED BY QUARTER**



## CONCLUSION

Examining a vendor's test documentation may result in over reliance on the documentation rather than the content of testing (or lack thereof). 'The assumption is that a documented process equates to quality, but this is a fallacy.' (McDowal[6]).

An aggregation of multiple test levels (unit to acceptance) and techniques (to be discussed in Part Two of this article) can significantly enhance the software product quality and reduce the risk of operational issues.

By pivoting focus from looking purely at testing documentation, to the test strategy and test measurement can establish the level of software product quality to better inform stakeholders.

This testing series will include discussions on aspects of comprehensive testing, including test objectives, test techniques, test process, test management, measurement and test tools. The authors aim to provide more information on the benefits of these points on software product quality (and compliance) and what to look for during vendor engagements. Each article will finish with one or more vendor assessment hints.

## REFERENCES


1. Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 4.0, IEEE Computer Society. 2024
2. Computer Software Assurance for Production and Quality System Software, FDA, 13th September 2022 (draft guidance).
3. Empowerment Quality Engineering Ltd
4. ICH guideline Q9 (R1) on quality risk management, EMA, 03 February, 2023
5. Kelvin, Electrical Units of Measure, PLA vol 1. Electrical units of measurement 1883-05-03
6. Bob McDowall, Computer Software Assurance: Perfect Solution or Confidence Trick? Technology Networks, 15 November, 2024


## PROFILES

Barry is a Principal Consultant for Empowerment Quality Engineering ltd, a Computerised System Regulatory consultancy that bridges the gap between IT and quality.

He focuses on building quality and security into Computerised Systems (CS) by using quality techniques from the wider software industry while ensuring regulatory compliance. He leads GxP CSV compliance and IT Supplier/ Service Provider audits across the globe; performs IT supplier's software lifecycle process improvement, risk assessments to drive validation strategies, validation projects and tailored training.

Barry has over 27 years' experience in Quality Assurance, Software Engineering and IT Administration with vast technical knowledge of every role and every activity within the CS lifecycle; including multiple technologies, development methodologies (traditional and agile), databases and programming languages.

He is a member of the RQA IT Committee, the MARSQA and was a member of the ISPE Data Integrity Project team.

Hugh is VP Operations and Quality at PHARMASEAL International Ltd and an independent computer systems validation consultant.

He is an IT professional with over 35 years of experience of using technology in the pharmaceutical industry, initially as a developer, later an implementer and more recently specialising in compliance.