

Agility in Medical Platform Testing – Lessons and Experiences of Seven Years

Dr. Sukanta Bhatt and Mr. Kishore Vinod
Philips Electronics India Ltd,
Bangalore, India
{sukanta.bhatt, kishore.vinod}@philips.com

Abstract:

This paper describes the challenges of Medical Imaging Platform Testing in Philips Healthcare group. We've encountered different situations the last seven years and gained sufficient knowledge, testing the whole software life cycle. We aim to present our learning from requirement agilities, re-factoring projects, demands of a regulated world, long sustenance periods of the medical platforms.

In addition, demands and usage diversities of different product groups who depend on the medical platforms are discussed

Keywords:

Testing, Lessons, Agility, Requirements, Platforms, Test Automation, Re-factoring, Sustenance.

1 Introduction

During the past decade of development, health-care modalities have become software intensive, particularly in the arena of medical imaging systems, diagnostic applications, and archiving systems. The demand for high quality and price pressure is driving healthcare industry to be more efficient and deliver products faster.

This paper aims to share our years of experience and challenges of medical imaging platforms testing at Philips Healthcare. The events spread across the Software Testing Life Cycle (STLC) starting from requirements to sustenance.

2 Agility of Requirements

Late requirements scope change in the SDLC (software development life cycle) is a fact of life. Sometimes a competitor new release with differentiating features demands the current development project to enhance the features as well. At times availability of new technology in between the project development brings in new value opportunity and efficiency leading to demand for requirement change. New insight into the customer need comes now and then after a few iterations leading to agility in requirements. In addition there are situations when the team understands the constraints of a design or technology choice during early NFR (non functional requirements) testing leading to change request. Most often than not, the impact on testing is a after thought.

Due to promise and correct entry timing in the market schedule slip had to be contained. A typical question the test manager meets – “how much you can absorb within the schedule?” Invariably creative explanations and estimations are provided by other stakeholders to the V&V (Verification and Validation) team to reduce lead time. Many a times there is a simplistic perception of V&V activity thinking -- it is only effort and capacity issue. However, any new requirement demands thorough analysis,

understanding, discussion, test estimation, specification and environment preparation and finally execution. These events have big impact on the STLC (software test life cycle) activities as little time are left for a quality preparation. We have had our share of requirement agility over last seven years and learned a few tricks.

For upcoming technology, keep an eye on various technology specific web journals and tech talks. It could be a service pack for the OS or database. There is a possibility of a new version as well which needs effort intensive environment preparation and specific impact testing. There are third parties applications need to be checked if they work with these new releases which could be difficult to resolve.

For early alert system for requirement change requests for technology and design constraints, test team need to be more innovative. Besides the formal project status meetings, we have also learned other means which are effective. The requirements repository server weekly scanning is a very good practice. Many a times, it is late scoping of the item though the request was already there in the database. At times technical decisions happen in architects meetings where as project management act a little later. Keeping track of technical MOM helps to understand the intricacies and prepare early. And finally, informal coffee table chat with senior developers brings invaluable insight about the upcoming changes and its impact.

3 Re-factoring

In software engineering, "refactoring" source code means modifying it without changing its behavior, and is sometimes informally referred to as "cleaning it up". Refactoring neither fixes bugs nor adds new functionality, though it might precede either activity. Rather it improves the understandability of the code and changes its internal structure and design, and removes dead code, to make it easier to comprehend, more maintainable and amenable to change. Refactoring is usually motivated by the difficulty of adding new functionality to a program or fixing a bug in it. [wiki]

Any major software system goes through a re-factoring in about every three years. At times it is required for design simplification and effective re-use. In addition new technology and COTS solutions also make a lot of old implementation redundant or look expensive. In big systems, package dependency minimization is a uphill task as most people are focused on functional aspects. In addition code hygiene is another reason for re-factoring. However one theme remains for all types of re-factoring – no change in functional behavior. Every re-factoring project has a key theme.

Since these projects do not create perceived market value, management is very cost sensitive and developers are casual at times. One usually hears remarks like, “here is no functional change”, “only cosmetic change in package structure”, “minimal testing shall suffice”, “sub-contract and run as a separate project”.

The impact (functional and NFR) of re-factoring is very hard to visualize as most of these changes apparently do not change the behavior. Documentation for these type of changes is mostly nonexistent. Technical decisions happen over white board sessions where most

of the testing people are usually not present. Many a times, re-factoring leads to some functional regression and NFR impact.

One needs a good automated regression test suite for re-factoring projects. This helps to manage cost and schedule. The testers need to understand the theme of re-factoring for effective test strategy. Since the documentation won't be extensive, channels need to be opened for critical information flow. Re-factoring overview from the key technical people at various stages helps a lot to focus on real impact areas. During these sessions they provide very valuable hints about the changes and likely problem areas. In addition, short incremental tests can be done early to identify the breaking features. Additionally, executing old test set with new configuration, hardware and data do reveal new defects.

4 Compliance World

Most of the system software comes under some regulatory and statutory requirements. Every domain and vertical has its own share of compliance needs. Healthcare is a highly regulated industry with many compliance standards and procedures which also vary per country. The knowledge of various compliances and standards and their testing adequacies play a major role in product releases. In the healthcare domain there are DICOM, IHE, HIPAA etc.

Digital Imaging and Communications in Medicine (DICOM) is a standard for handling, storing, printing, and transmitting information in medical imaging. It includes a file format definition and a network communications protocol. The communication protocol is an application protocol that uses TCP/IP to communicate between systems. [wiki]

IHE is an initiative by healthcare professionals and industry to improve the way computer systems in healthcare share information. In 1997, a consortium of radiologists and information technology experts formed IHE, or "Integrating the Healthcare Enterprise." IHE aims to create a process through which interoperability can be implemented.[wiki]

The Health Insurance Portability and Accountability Act (HIPAA) was enacted by the U.S. Congress in 1996. According to the Centers for Medicare and Medicaid Services (CMS) website, Title I of HIPAA protects health insurance coverage for workers and their families when they change or lose their jobs. Title II of HIPAA, known as the Administrative Simplification (AS) provisions, requires the establishment of national standards for electronic health care transactions and national identifiers for providers, health insurance plans, and employers.[wiki]

Compliance requirements are a specialization of it own which needs training and experience. Vendors also provide tool set for various compliance requirements. Adequate training needs must be planned for test team and these tools. In addition the application context know-how is critical to effectively evaluate for compliance. Visit to application sites and sessions from specialists to enhance domain knowledge is imperative to do effective testing.

4 Longer Sustenance

Healthcare products are complex and expensive in terms of money and efforts. Hospitals take years to cover the investment. The sustenance period is long compared to other products. The software stacks in the healthcare platform are re-used for different releases. A common perception of maintenance is that it is merely fixing bugs. However, studies and surveys over the years have indicated that the majority, over 80%, of the maintenance effort is used for non-corrective actions (Pigosky 1997) [Wiki]. Also maintenance is really evolutionary developments (Lehman 1997).

Maintenance projects always have high release and capacity pressure. In addition there is lot knowledge of user issues and how the software behaves in the field. However this mostly remains within the team and is lost as people keep of changing in these projects. For effective testing, it is absolutely necessary to understand this knowledge and a good rapport with the development team.

Field defects are fixed in the released products as well as in new versions. In such cases, a seemingly trivial fix in an older release can manifest as a major bug in the higher releases due to design change, re-factoring or environment change. These types of defects are hard to predict. Insight into the structure of the higher archives and changes are critical to do a good testing for these defects.

It is ever so important to keep an eye on code changes in the older release for the software stack. This can help in an effective grey box testing which is very efficient for bug detection. In addition good low level automated regression test coverage comes as a life saver many a times. Special tools can also be used to check change impact.

4 Test Automation

Most successful automators use a software engineering approach, and as such most serious test automation is undertaken by people with development experience [wiki].

The major impact on testing is felt when the manual test efforts far exceed the demanded delivery cycles. However test automation is not an easy talk though many a times it starts with an easy perception only to end up with frustrations.

Early identification and planning of automation as a phase in testing is critical because of the fact that if the efforts to automate are realized, major pains can be avoided during the releases and maintenances later. For sustainable test automation, it is important to treat as a development project with all the engineering practice. Involving the development team for low level automation is highly beneficial as they bring insight of the design. Also, it is necessary to use a variety of automation tools that cater to different facets of testing from unit-testing to the system-testing levels and also from line tests to the UI tests. Suage of a test framework makes the scripts clean and structured. Owner for various scripts shall be identified upfront so that failure can be fixed systematically. Involvement of senior development members in test framework design leads to a mature and stable test suite. It is a good practice to integrate the automated scripts with the build process, In addition graphical results shall be published regularly to show the value from automation.

References

- [1] Cooper, Victor: 2008. *Noun & Verb Technique*. New York: Pure Publications.
- [2] Fowler, Martin (1999). *Refactoring. Improving the Design of Existing Code*. Addison-Wesley. [describes 72 specific "refactorings"]
- [3] Pigosky T.M. (1996). *Practical Software Maintenance*. New York: John Wiley & Sons.
- [4] Lehman & Belady (1985). *Program Evolution*. London: Academic Press Inc.
- [5] Elfriede Dustin, et al: *Automated Software Testing*. Addison Wesley, 1999.