Requirements, Architecture, and Quality in a Mission Critical System: 12 Lessons Learned

Aapo Koski
Insta DefSec Oy
Tampere, Finland
aapo.koski@insta.fi

Tommi Mikkonen
Tampere University of Technology
Tampere, Finland
tommi.mikkonen@tut.fi

ABSTRACT
Public tender processes typically start with a comprehensive specification phase, where representatives of the eventual owner of the system, usually together with a hired group of consultants, spend a considerable amount of time to determine the needs of the owner. For the company that implements the system, this setup introduces two major challenges: (1) the written down requirements can never truly describe to a person, at least to one external to the specification process, the true intent behind the requirement; (2) the vision of the future system, stemming from the original idea, will change during the specification process – over time simultaneously invalidating at least some of the requirements. This paper reflects the experiences encountered in a large-scale mission critical information system – ERICA, an information system for the emergency services in Finland – regarding design, implementation, and deployment. Based on the experiences we propose more dynamic ways of system specification, leading to simpler design, implementation, and deployment phases and finally to a better perceived quality.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *Elicitation methods, Methodologies*.

General Terms
Design, Human Factors, Verification.

Keywords
Vision; requirements; user stories; architecture; external quality; internal quality; process quality.

1. INTRODUCTION
Public tender processes of information systems – almost regardless of the target and characteristics of the system itself – start with a comprehensive specification phase. During this phase, the representatives of the future owner of the system, typically together with hired consultants, spend a considerable amount of time to precisely define the exact needs the owner has in mind for the system. Once requirements are fixed, it is up to developers, usually employees of another company, to create a design to meet the requirements.

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2804436

In the agile era, where interactions between developers and other stakeholders are advocated, this traditional view is challenged. Characteristics other than implementing requirements only are to be considered, which calls for improved setup as well as mindset for the development. In general, neither the customer nor the developing organization has an off-the-shelf solution. Rather, joint work is required to find functional practices for cooperation.

In this paper, we present real-life experiences based on ERICA, an information system for the emergency services in Finland, serving the emergency call-taking and dispatching emergency tasks to police, ambulances, fire department, border guard and social services, made available to end users as a service. ERICA is large, multi-million euro construction. The presented experiences have been gathered over several years, and they cover various aspects, such as project and requirement management, architectural design, development process improvement, quality management, and end-user involvement.

2. NEEDS AND REQUIREMENTS
Proper requirements are essential ingredients in producing high-quality software. These define, one by one, a specification that stems from the original vision of the needed solution.

In the specification phase, the people involved spend a considerable effort and a period of time thinking thoroughly the exact needs the future system must satisfy. For instance, in the ERICA project, the customer started specification of the project more than two years prior to the start of the procurement phase, resulting in a pile of documentation describing the system from many points of view, but having a strong emphasis on the functional aspects. The specification produced lists of hundreds of requirements, written carefully in IEEE 830 requirement format [1]. These requirements were reviewed over and over, they were amended and refined, some of them got rejected and new ones were introduced. Finally a set of requirements were accepted as the basis for the development.

Accepted requirements serve as the most appreciated artifact for the following steps in the information system's design and development – the ultimate "truth" to which one can refer to when debating system features or characteristics and by which the customer eventually assesses the progress and the quality of the system. They are offered to potential vendors for estimating the time and cost to implement a system that could perform or behave as stated by the requirements. Retrospectively, for our project it would have been valuable, instead of just getting the list of final requirements, to be able also to see the process leading to the final requirements, including older versions and rejected ones.

When specifying the requirements as described above, we make – partly consciously, partly unconsciously – some assumptions. For instance, we assume that the representatives set to specify the system have all the knowledge required to do the job and that they

are capable of expressing the needs in a written format that cannot be understood in a wrong or incomplete way and leave as little space as possible for vagueness. We also assume that the understanding on the needs we have will not change significantly during the specification process. We further assume that the requirements can be written in the same way regardless of whether a requirement specifies a functional feature or a quality issue and that the requirements are expressed in such detail that the potential vendors can reliably estimate the time and costs required to implement each requirement. Finally, since information systems are nowadays created in an agile – although a more appropriate term in public tender projects is agilish – way, the requirements should be written in a way that allows incremental development. Ideally, the requirements are completely independent of each other and can therefore be implemented in any order.

As anyone involved in information system design and development can verify, the fixed set of requirements is already a problem in itself. If not adjusted and refined in a continuous manner, the requirements will lead to an implementation that does not correspond to the actual needs of the system owner. Moreover, even if the requirements itself were complete and reflect the needs of the system owner, the form and the language used in the writing of the requirements often leaves a lot of room for interpretation. This problem is emphasized when the project is large in size, since the same requirement are interpreted differently in different phases of the project, when scrutinized by different people playing different roles in the development.

When considering the origin of requirements, three distinct sources can be identified: 1) system sponsors, defining the business, integration with other systems, security, schedule, and effort or cost; 2) system users, specifying the required functionality, usability, availability, learnability, performance, and administration; and 3) authorities, defining laws, regulations, contractual obligations, industry standards, and applicable practices. Reflecting these sources to the ERICA system development, the system users were obviously the ones who best got their voice heard, whereas system sponsors' and authorities' opinions were not taken into account in full.

Lesson #1: Even if the client and the contractor both wish to execute a project in an agile fashion, joint practices and processes must be carefully agreed upon.

Lesson #2: The stakeholder that the developer hears best ("the loudest voice") may not be the most important stakeholder.

Lesson #3: Feature management system is not a replacement for interactions with end-users and other stakeholders.

3. SPECIFYING REQUIREMENTS

Capturing requirements is in general difficult; however, capturing significant requirements for a mission critical system is particularly difficult. A critical system is a system whose failure could threaten human life, the system's environment or the existence of the organization which operates the system. The concept of criticality introduces special needs on the way the system is specified, planned, designed, implemented, tested and finally deployed to use. It is important to notice that failure in this context does not mean just a failure to conform to a specification. Rather, such failure means any potentially threatening system behavior, be it within specified functions or not. This fact puts the developers of mission critical systems in a very different position than the developers of less critical software when considering the design and eventual acceptance of the system into production. In particular, conformance to a set of requirements is not enough.

For better or worse, agile practices have also become a part of building mission critical information systems (see e.g. [2], [3]). In contrast to mission criticality, agility in our mind should mean only three things [4]: user involvement, iterative and incremental development and constant adaptation to the situation at hand.

To ensure the preservation of dependability throughout the development, special attention must be placed on mission criticality in the design of the system itself as well as tools, techniques, and processes that are used during the implementation as well as testing and other validation activities.

Finally, for an end-user of a mission critical system, criticality does not only mean that system functionalities are up and responding swiftly. It is also critical that the user interfaces support the user in executing her tasks and make her as effective and informed as possible. Consequently, the developers must understand the characteristics and the behavior of the end-users. Understanding regarding these emerges gradually when developers and end-users co-operate.

In the ERICA project, due to its size and length of the project, we encountered many occasions where the original requirements alone would have resulted in an incomplete or unreliable solution. We also had to do many changes due to regulations and laws that for some reason were not fully taken into account in the specification phase.

Since the project was executed in a close co-operation with the end-users, we received a lot of feedback on the functional quality aspects. When doing this, we were not careful enough to avoid the feature creep. However, the responsibility on keeping the scope set by the system owner should not be on the development teams but on the customer representatives and end-users. It seems to be somehow many times forgotten that keeping the feature creep under control is of the highest interest for all stakeholders. In case feature creep takes place, the whole project is immediately in danger and in the worst case no system will be ever created.

Lesson #4: New features are easy to invent; each feature should be associated with explicit stakeholder value.

Lesson #5: To avoid feature creep, one needs to perform rigorous and visible change management. Learning to say "*No*" in a nice way to end-users is obligatory.

Lesson #6: Criticality is a part of every requirement, not a part that can be isolated.

4. ARCHITECTURE DESIGN

Architecture in a system can be defined as everything that is hard to change. Architecture is the "load-bearing walls" around which the rest of the software can be designed [5]. Consequently, the architecture cannot stem from the same requirements that define the functionality, as those requirements usually evolve over time.

When the requirements are defined in the fashion described earlier, the architecture of the system typically plays only a minor role or no role at all in the process. This is, of course, understandable – the people involved in the specification are mostly concerned on the functional features of the system to be

developed, since that is what these people are requested when they interact with end-users and other stakeholders. Although some of the requirements can be categorized as architecturally significant requirements, the main features and required characteristics of the system architecture remain largely unspecified and vague. Especially now at the agile era, where room is typically left for later fine-tuning and revisions, requirements that are specified up-front in the project lead easily to a so-called emerging architecture. Emergent architecture in itself sounds acceptable – and indeed is acceptable for various projects – but for long-lasting systems, such an approach means that the system architecture has to be rethought many times during the project. This in turn costs money, makes the system development slower, limits the options, forces one to make sub-optimal compromises and brings about a lot of confusion.

It has long been recognized that system architecture has a strong influence over the life cycle of a system [6]. However, until relatively recently, hardware issues have tended to dominate architectural thinking and software related aspects of architecture – if considered at all in the first place – have often been the first thing to be compromised under the pressures of development.

In ERICA, the original sketches of the architecture and the final design surely resemble each other. However, the reason for this is not comprehensive pre-implementation specification phase, but merely high cost of architectural changes. Due to the time, amount of work, and schedule related to the needed changes, we have spent a lot of time to finding ways to bend the architecture to make at least some minor mandatory changes possible. At the same time, some original requirements, which in themselves are fully justified but not easily implemented with the current architecture, were interpreted in a way better compliant with it.

Lesson #7: A verifiably satisfying architecture for a mission critical system simply does not emerge; it must be explicitly designed and validated.

Lesson #8: Architecture is not a concern of any of the representatives of the system's owner. Therefore, the developers should be extremely concerned on its design.

Lesson #9: Limits of the architecture will be met in any case; create a strategy for managing overarching requirements.

5. QUALITY

Quality is a topic of great interest to all stakeholders involved with an information system, including also the developers and end-users. Moreover, quality is what the system owner and all stakeholders are really looking for, instead of fulfillment of requirements. To avoid mismatches and false expectations, the definition of the term quality should be the same on both sides of the table when negotiating and discussing with the customer. Moreover, the same definition must be applicable also at the point of delivery and deployment.

There is no lack of definitions or specifications regarding what quality means (see e.g. [7]) and from which points-of-view the quality of a system should be assessed. In addition to external quality issues, comprising of both functional and non-functional issues, also the internal quality (which reflect e.g. the testability and the supportability of the system) and the process quality (which ensures the project is on time and on budget) needs to be taken seriously into account when assessing the overall quality.

The functional quality should be straightforward to measure, since if we have a representative of an end-user at hand, we can provide her the system as it is, tell her what features are already implemented, how the features can be used and ask her, is she seeing and experiencing by the use of the system such things that qualify as the final features delivered to the production. Unfortunately, this kind of behavior did not take place in ERICA project and rarely does. Instead, we far more typically find that instead of getting feedback on what are the most important issues missing from the system, we get masses of defect reports, ranging from irrelevant comments on the look or behavior of the user interface to reported bugs on features not in scope.

Lesson #10: Defect reports are no replacement for true interaction with representative end users.

Lesson #11: Test automation is a necessity for agile development.

Lesson #12: Quality in the large cannot be validated with technical tests only; instead, end-user involvement is needed.

6. DISCUSSION

Based on the above lessons, it is obvious that when doing the classic waterfall-style [8] system development, the system owner and other stakeholders have no control over quality. In contrast, when playing by the rules of agile development, the playground the requirements, the system architecture, the quality and the acceptance testing form a diagram depicted in Figure 1.

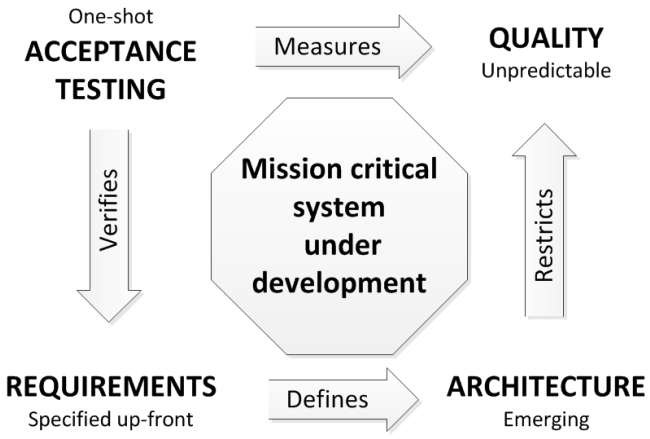

Figure 1. Roles and relations in an "agilish" approach.

Even doing the project in an incremental and iterative way, the system owner still does not have any guarantee that the provided system, with fixed time and cost, will converge to what is actually needed within the budgeted time and money. The achieved quality is more or less a surprise – sometimes possibly a positive one but typically less than was expected – and restricted by the emergently created architecture. Taking into account these problems, the development should preferably follow the enhanced process presented in Figure 2.

The project needs to start with some clear quality needs, consisting not only of functional quality aspects, but covering all the quality aspects the system owner and her representatives have

in mind. The format of the quality need should be that of a user story, stating who needs something and why. The system architecture should build on these quality needs and reflect the up-to-date understanding of the required system quality.

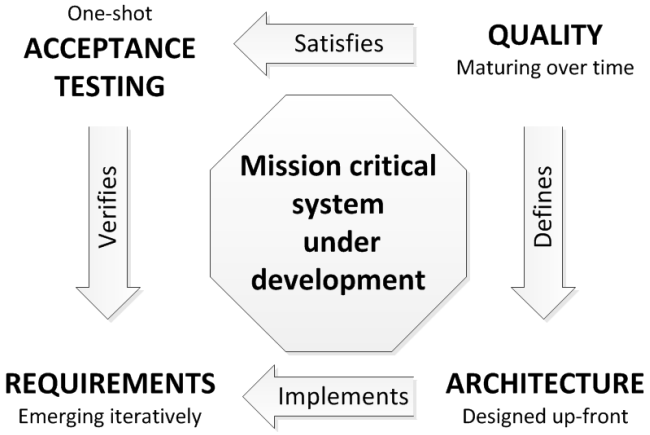

Figure 2. Roles and relations as they should be.

The acceptance testing should be done against the required quality aspects as they are at the moment of testing. The testing should concentrate on the validation of the implemented quality aspects and give a lot of feedback on all tested quality features, guiding the next development and testing efforts. The requirements – if required for example for contractual reasons – can be defined and written on the basis of the implemented quality and the results from the acceptance testing. Important aspect is that the requirements here emerge iteratively and allow us to monitor and analyze the way the system is evolving.

Finally, the requirements, the architecture, and the original needs or vision of the system to be developed and taken into use give us three points-of-view into the system. These views are not necessarily similar views, however. Ideally, these three elements, complemented with the acceptance testing criteria or test cases, should support and reinforce each other and together give any stakeholder a clear view on what are we developing, how the development is progressing and how well it satisfies the needs we have.

7. CONCLUSIONS

The era of the development model where we try to specify the information systems by writing down a set of requirements in short format has lasted for a long time. During that time it has been proven over and over again that by such process we do not end up with results that satisfy the system owners; we either end up providing a system that does not include the features really needed, or we do not converge in the development effort to the right system solution within budgeted time and budgeted cost. This fact is emphasized with mission critical systems because of their special nature – they just simply cannot behave is a way that causes threat to human life or property, no matter the requirements are fulfilled or not.

In general, functional quality specifications should be written by someone who is not involved in any other aspect of the project developing the system. The writer should be familiar with user interface issues and web design (if required), familiar enough with the used technology to know the possible limitations and capabilities, and someone who is a very skilled communicator and definitely a good writer. While writing a specification, the writer should spend a great deal of time imagining how a user might use a certain feature and how they may navigate their way through the software. The functional specification writers' main concern should be marrying the user experience with the various business logic requirements of the project.

The non-functional quality specifications should be also written by someone who is not involved in any other aspect of the development project. This time, however, the writer should be fully aware of the required system capabilities, acceptable performance criteria, number of users, events and transactions, the needs for scaling the system and of any limitations set to the system development, like the characteristics of the environment or existing regulations or laws.

To conclude, we strongly feel that the system owners are not to blame for the problems encountered in large-scale information system development projects. It is the people who design, develop, test and deploy the system who are responsible to educate the system owners how high quality information systems should be put together. Starting from quality needs instead of requirements enables right solutions to the true needs, longer system lifespan, and eventually happier customers.

8. ACKNOWLEDGMENTS

We thank Insta DefSec Oy for the opportunity to report on the experiences gained in this large-scale mission critical project.

9. REFERENCES

[1] IEEE Std 830-1998 (Revision of IEEE Std 830-1993), IEEE Recommended Practice for Software Requirements Specifications.

[2] Bowers, J., May, J., Melander, E., Baarrman, M., and Ayoob, A. "Tailoring XP for large system mission critical software development." *Extreme Programming and Agile Methods—XP/Agile Universe 2002.* Springer Berlin Heidelberg, 2002. 100-111.

[3] Drobka, J., Noftz, D. and Raghu. R. "Piloting XP on four mission critical projects." *IEEE Software*, 21.6 (2004): 70-75.

[4] Koski, A. and Mikkonen, T. “Rolling out a mission critical system in an agilish way,” In Proceedings of 2nd International Workshop on Rapid Continuous Software Engineering, May 2015.

[5] Perry, D. E., and Wolf, A. "Foundations for the study of software architecture." *ACM SIGSOFT Software Engineering Notes* 17.4 (1992): 40-52.

[6] Gacek, C. Abd-Allah, A., Clark, B., and Boehm, B.. "On the definition of software system architecture." *Proceedings of the First International Workshop on Architectures for Software Systems.* Seattle, Washington, USA 1995.

[7] ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.

[8] Royce, W. "Managing the development of large software systems." *Proceedings of IEEE WESCON*. Vol. 26. No. 8. 1970.