XP 2017 COLOGNE

# Taming a Monster: Tackling the Emergent Issues Encountered in Mission Critical System Development

AAPO KOSKI, Patria Oyj
TOMMI MIKKONEN, University of Helsinki

Many large IT systems have become too complex to understand. The complexity stems from the ever-increasing number of technical parts as well as from the increasing number of people involved. Still, the systems are designed, developed, tested, and taken into use without paying too much attention to the issues the inherent system complexity introduces. These issues – called emergent because we are not able to foresee them – result in system failures, downtime, and hasty workarounds and fixes. These, in turn, lead to unsatisfying user experience and unexpected costs. This paper is an experience report on some problems encountered and on the solutions applied during a mission critical SaaS system project, especially from the system testing and monitoring standpoint. Main lessons learned, regarding the ability to control and understand the mission critical service behavior, are that masked production data is a must for performance testing, and that the test cases should be automated early and not developed further and enhanced all the time. One should also remember to trust the performance test results, once assured that the environment has been steady.

## 1. INTRODUCTION

Everyone dealing with large IT systems readily admits that systems have become almost too complex to understand (Lyu, 1996). They are still often designed, developed, tested, and taken into use without paying enough attention to the issues the inherent system complexity introduces. These issues – often called emergent – result in system failures, downtime, and hasty workarounds. These, in turn, lead to unsatisfying user experience and unexpected costs.

When dealing with a mission critical information system, failures and downtime are unacceptable, to the extent that it is not an option to even risk that the users start to doubt reliability. Therefore, a mission critical system should always operate in a way that the users can rely on. The users need to be able to trust that almost whatever happens, the system will be available to the users in the best possible fashion, and help the users to perform tasks to the extent the present situation allows. In addition, the user must always feel confident with the system. Consequently, the complex mission critical system – the monster of this story – needs to be tamed appropriately and with proper tools and processes.

When a mission critical system is first built, be it out of licensed or custom built software, then taken to use, and finally maintained by an organization other than the one that has developed and deployed it, the challenge of divided responsibilities emerges (Liu, 2016). In the presence of any event regarded as a problem, it is not imminently clear which party is best capable of tracing the bug or who is responsible for providing a fix. Furthermore, time, being always a scarce resource when dealing with mission criticality, is wasted when the different organizations are spuriously communicating to find a solution to the problem, or, even worse, to just understand which party should react to the problem at hand.

Considering responsibilities, the software-as-a-service (SaaS) (Turner, 2003) model is a good fit with mission criticality in several ways. When the system is provided as a service by one service provider, there is no doubt who to blame in the occurrence of errors, who to contact as problems arise, or who should take care of observed problems or issues raised by the users – the provider always bears the responsibility. Also, the potential problems with communications should be less likely when the communication takes place within one organization, not between several ones. However, at the same time, the SaaS model puts a totally different kind of burden on the shoulders of the company providing the mission critical service. If the service has in practice no option to fail, the system design, development, testing, deployment, and monitoring while being operational, must be done with the mission criticality as the first thing in mind. One of the most important aspects is the thorough understanding of the provided service characteristics: how the system behaves in ideal environment, how the behavior changes when the environment changes in some way, what are the limitations of the system, and what kind of anomalies we should be monitoring to be able to detect faults and possibly resulting errors.

Author's address: Aapo Koski, Patria Oyj, C4ISTAR, Hatanpään valtatie 30, 33100 Tampere, Finland; aapo.koski@patria.fi
Second author's address: Tommi Mikkonen, Dep. of Computer Science, 00014 University of Helsinki, Finland; tommi.mikkonen@helsinki.fi

In this paper, we report experiences gathered during the development and deployment of a mission critical information system, used by security authorities. In accordance to best agile practices, the system was developed in close cooperation with the users in an iterative and incremental way. Due to its nature, the system needs to have an extremely high availability and robustness. Such high availability means that in practice no downtime can be tolerated and downtime periods for maintenance of any kind are not available.

## 2. ABOUT THE MISSION CRITICAL SYSTEM

A mission critical system is a system that is so essential to the survival of an organization using it that a failure of the system or an interruption in the system functionality impacts business operations significantly (Mission critical, 2017). In many cases, a failure may cause a threat to human life or at very least cause significant economic losses. Therefore, to put it simply, the downtime of such a system is not tolerable during the system's specified operational intervals. For public security systems – the domain from which views and experiences discussed in this paper have been collected – the specified operational interval is 24 hours per day, all year around.

The system our experience stems from is built for operational use by hundreds of simultaneous users and has strict performance and reliability requirements. The system is provided as a service for the authorities responsible for the emergency response functions nationwide (New national ERC information system, 2017). The service is provided as a service by a private company and it is targeted to be run at an availability level of 99.996%. The system comprises of several COTS components as well as custom tailored software modules and relies in its operation on many integrated systems with varying integration capabilities and dynamics.

The major part of the system was designed and developed during a period of 5 years with a team of up to 30-40 persons. Being a SaaS system, the development effort naturally continued also after the first versions of the system were deployed for customer testing and approval for operational use. The organization of the development project comprised of 4-5 cross-functional agile teams which used Scrum and worked on team-specific backlogs and in time-boxed intervals, typically in sprints of 3-4 weeks. Testing was performed by the teams as new features were developed. In addition, we had an operations team responsible for integration and system testing as well as for deploying the released versions of the system for customer testing and use.

The first author of this report has been the chief architect of this system from the very beginning of the project and was responsible for a large part of the technical aspects of the project, ranging from the refinement of the system requirements with the customer to the ramp-up of the development teams and specification and building of the required system environments and testing facilities.

## 3. INFORMATION SYSTEM AND ITS ENVIRONMENT

Large-scale information systems, which contain million lines-of-code or more, easily become so complex that their behavior is next to impossible to comprehend. This complexity stems from the fact that such system comprises of many components, which introduce emergent behaviors when acting dynamically together, either in foreseen or in unforeseen ways. In addition to system components, the system-level behavior is also dependent on the state of the environment as interpreted by the system, and this environment can be even more complex than the system itself, resulting in further emergent behaviors. What is obtrusive in this context is that in isolation the characteristics of the components are usually well known and documented. This easily leads to a false assumption that large-scale information systems can be developed, tested and taken into use without paying much attention to the issues the inherent system complexity and resulting emergent behaviors introduce.

Failing to pay enough attention to system complexity and the interaction between the system and its environment result in infamous system failures and unwanted downtime. When problems are encountered, hasty workarounds are often introduced, which often make the overall situation even worse. Failing to foresee operational system behavior causes unsatisfying user experience and unexpected costs, both of which the software industry is already much too famous for.

**Similarity Between Test and Production Environments:** To enable pushing complex systems to production environments, several intermediate environments are needed. Some are used for development, some for testing and training, and finally at least one for deployment. While the most interesting environment, both from users' and developers' point of view, is the operational environment where the real action takes place, the others must be consistent with the operational environment for the development and testing efforts to make truly sense. Otherwise the results we get from the testing – both in terms of technology and user experience - do not correspond to the ones obtained from the real operational system.

The relevant questions here include how well we need to know the environment and its characteristics, and how much is enough when considering similarity between the installation environments. If we cannot even tell if two environments are different or not, we are already losing the game of controlling a running system installation. Furthermore, to add further complexity, it is just not the static configurations of the environments we must deal with, but all environments are dynamic and constantly experience changes. The changes taking place are either caused by intentional actions performed by the many stakeholders involved, or by unintentional or unexpected incidents of varying nature such as operating system updates, networking conditions, and so on.

To summarize, the criticality of various actions and possible incidents need to be considered, and their consequences need to be known. In addition, we must be able to observe changes in the environment's characteristics, and have the capacity to react accordingly. In our cases, we started from an early phase to release and test the developed parts of the system together in several environments that resembled as closely as possible to the future operational environment as we understood it. We had a separate identical environment for the customer acceptance testing and for the performance, stress and load testing. The setup and maintenance of these environments, although heavily virtualized, requires a notable amount of work and therefore specific resources assigned to the maintenance and control of these environments solely.

**Controlling the Environments:** Without appropriate control over the environments used to run the system, there is no way to protect the users from emergent events that are imminent in complex systems. The need to know what has been done and what is planned to be done to the environment is of the utmost importance to understand the consequences of a certain change. Still, control does not mean that we prevent the changes from taking place or restrict the possible changes that are allowed. Any modern information system must be built to allow flexibility for varying, ever changing needs of the users.

In our cases, although the environment configurations were documented in a very detailed way, one of the problems encountered was how to ensure as automatically as possible that the documentation corresponds to the real situation. It was more than once that the human factor surfaced: once a configuration problem is solved in one environment after possibly a notable effort of trials and errors, it seems to be relatively hard to remember to document the needed change appropriately and to remember to do the same change into all the other related environments having the same system configuration.

With the SaaS model, unlike traditional system provision models, the service providers typically can configure and control the system environment as needed and thus the above points should not be a problem.

## 4. ASSESSING MISSION CRITICALITY

With mission critical information systems, failures and downtime cannot be tolerated in the same way or at the same level as with non-mission critical systems. A mission critical service should always run so that its users can fully rely on it and trust that no matter what happens, the mission critical system remains available. This calls for mechanisms that help detecting problems as well as recovery mechanisms when problems have been identified, i.e. tools with which we can monitor and control the monster we have created. As with the environment, specifying, setting up, and using various mechanisms for taming the system with SaaS model depends only on the skills and competencies of the system provider.

### 4.1 Monitoring Mechanisms

Monitoring a critical service is something one does not want to do with tens of different monitoring management products. The management system of the monitoring solution should also be able to report on performance levels, as that will be a key aspect of the service level agreement (SLA) or equivalent availability indicator of the service. The SLA's the customers require need to be met without having to double computing infrastructure or having to add separate staff to implement High Availability (HA) and Disaster Recovery (DR). This need for comprehensive tools calls for developers with a true DevOps mindset. We started by having the monitoring solution and the service itself in separate silos managed by separate people which resulted in problems in communication, problem solving quality and above all, the response times to solving the observed problems. To get rid of these problems we had to find people who were capable to understand both the system and the monitoring solution and willing to be responsible for the entire provided service, not just the system itself.

A factor that often seems to be forgotten when information systems are build, is that if we have any kind of monitoring capability to monitor and control the system or service, the monitoring system is as critical part of the system as the rest of the system (Landau, 1969). The monitoring solution thus needs to be designed with the same level of reliability and robustness as the system itself and the components of the monitoring solution need to be under the very same configuration management as the operative parts of the system. We may even

have to consider monitoring the monitoring solution to ensure that our monitoring system is operational and provides reliable information.

In our cases, we are monitoring the system on two different levels; reported state and deduced state. Reported state is the state that the service itself claims to be in. Each service in the system publishes its own availability, load and response time information via JMX (Java Management Extension, 2017) for any observers to see. Reported availability does not necessarily tell if the service is usable to a client, so the main purpose of this is to help pinpoint issues when they occur, deliver extra information about the services' internal states and construct an overview of component's availability. Deduced state is the service's state as seen by an interested party, and is measured by calling the service and seeing when, how and if it manages to respond. For this purpose, service APIs share a common, unified monitoring interface, which allows checking their state in a generic way. All client applications can, and should use this method to measure the availability, as displayed to users, since it represents the factual usability of the service.

The amount of work and energy required to design and implement a reliable monitoring solution for a mission critical system is huge and calls for special skills. Per our experiences, a reliable monitoring system cannot be built afterwards for a system. Instead, it must be developed and grown together with the system itself.

### 4.2 Redundancy Mechanisms

Redundancy can be used as a tool to build mission critical systems that are computation and communication compatible. Intuitively, redundancy is a perfect tool with which we can promote coordination of the provision of the service. Redundancy is used to check whether an individual component of the system offering the service has deviated and in case we observe some deviation the redundancy mechanism ensures that the deviated component is replaced by a fully functional one.

Problem with redundancy is that typically we cannot objectively determine what level of redundancy is needed and what kind of redundancy should be employed. Redundancy also further complicates the system – the redundancy mechanisms itself may become the weak links in our system with unforeseen or surprising characteristics (Dubey, 2007). Furthermore, building the redundancy is in most cases very costly, and testing features related to it is complex.

In our cases, we started with modeling the most critical process flows in the system. With the model, we sketched out all the possible failure scenarios we could think of, ensured that if such a scenario takes place we observe it and designed ways to tolerate these situations.

Finally, redundancy alone is not enough. Providing a service that requires 100% uptime requires the ability to know when there is a problem. Moreover, we also need to know that there is a problem likely to surface in the future. This puts a high pressure on the monitoring mechanisms to detect all the relevant problems.

### 4.3 Service Requirements and Criticality

Precise, covering requirements form the foundation for any project trying to provide good service to customers. All changes that come from the customer side need to be verified, and appropriate process for handling the changes needs to be established from the beginning. With the verification, also the understanding on what will be the effect on the testing needs to be clear.

We learned that to maintain the mission critical service under control – especially when it is developed in an agile fashion as ours were – the most important things related to the requirements are the following:

- *Cut the functionality:* We were far too optimistic in the beginning regarding the system scope and estimated time to develop it. To survive, we should have always looked for the simplest way to satisfy a user need. One needs to remember that satisfying a need does not often mean fulfilling the written requirement.
- *Control feature creep:* We were also much too eager to answer yes to nice-to-have features and wishes the customer or the end-users had. Trying to please the customer with this kind of behavior will, sooner or later, have catastrophic effect on the project schedule and budget. This needs to be fully understood by all players and on daily basis everyone involved should critically assess the work at hand – is this what I am doing providing value to the customer and are there more valuable issues that should be taken care of first?

### 4.4 Critical Configuration Management

In addition to the configuration management of the normal situation – so-called happy day scenario – one must do similar configuration management also for disaster scenarios, where the system configuration changes due to responses to some faults or otherwise exceptional situations (Dart, 1991). The same applies to recovery

plans, which in addition to being properly tested and proven ones, should be managed with similar configuration management than all other setups of the system in question.

We learned that the requiring smaller but more frequent updates to the system, emphasized in the SaaS model, does not make the situation easy with regards to the configuration management (Dubey, 2007). A disciplined mode of operation with new streamlined processes to handle the changes needed is a must-have.

## 5. UNDERSTANDING THE MONSTER

We test to ensure that the system or service behaves as it is expected to behave. Testing is used to have a measure on the user impact to the service in various situations and in various ways. By performing testing, we try to achieve the needed level of service quality and to do that we naturally must define precisely what is meant by the quality with regards to the service at hand (Canfora, 2009).

Traditionally, software quality has focused a lot on the number of defects in a system, and testing has been the main technique for decreasing those defects by first finding them and then enabling their fixes. However, with mission critical systems, quality is not something one can introduce into a service via testing – it must be part of the service from the very start. Testing a mission critical system has, however, even more into it than just ensuring that the observed quality corresponds to the expected quality. With criticality, the expected quality may not always be enough. A critical system behaving exactly as planned but resulting in economic losses or even losses of lives while performing some action or restricting some action not be performed for some reason, is not obviously in retrospect of good quality.

To achieve confidence on the service behavior, we must be sure that we understand the customer or end-user expectations right from the beginning of the development process, including not just the functionality, but also usability, reliability, supportability, and performance, as well as business logic involved.

### 5.1 Functional Verification

For mission critical system, the functional quality must be understood in a broad sense. Functional quality does not just mean that a certain function is available for certain user to perform some action but high functional quality should mean that the functionality is for the user to use in all needed situations, informs the user in appropriate way about any issues related to the functionality and in case the functionality is somehow not fully available, offers the user options how to act to achieve wanted results.

In our cases the size of the systems poses a challenge to functional verification. In fact, it was clear from the very beginning that the major part of the functional verification needs to be fully automated. This facility would also act as a setup for regression testing. To this end, we developed a model based testing setup where by configuration we could setup and run several models of end-users. These end-user models behave a lot like the real end-users with statistical variation on the response times and process actions and we can run as many end-user models simultaneously as we want.

Developing a system in incremental and iterative way forces the development of the tests to be incremental and iterative. A mistake that we did in our projects was that we didn't understand to stop developing the test cases further all the time. Changing the tests hides the real system characteristics and performance.

To cope with the enormous amount of work involved in testing, the full automation of the release and build test cycles should be enabled as early as possible. As it has been reported that even 60% of the total system cost is spent on different types of testing (Paré, 1997), the investment into test automation can be justified. Our experiences support the observation that although automation of test cases have a high implementation and maintenance costs, automation of test cases can give remarkable returns in the long run (Kumar, 2016).

Finally, although emphasis needs to be on the automated testing to have the time and resources, the manual testing has its place, too, however. For example, ad-hoc testing cannot be performed using automation and negative testing can be done more rigorously with manual testing (Stobie, 2009). However, based on our experiences, the skill-set required from such effective manual ad-hoc tester is far from the role of a traditional software tester. Good scripting skills, ability to think outside the box and true will to break the system under test seem to be a rare combination.

### 5.2 Testing for Quality

The importance of testing the quality of the created system from the very beginning cannot be emphasized enough. With mission criticality in the picture, the role of the testing of the non-functional issues is even more important. Naturally, testing for quality requires that we know what kind of quality is needed – a simple and

axiomatic thing, but extremely hard to get right. Table 1 lists the most serious problems encountered during the development of the system and the mitigation actions taken.

*Table 1. Pain points and mitigations in mission critical system testing*

| Pain point | Mitigation | Comment |
|---|---|---|
| We set up the testing environment in a relatively ad-hoc way, thinking roughly along the lines "better to have at least some kind of environment for the testing than nothing at all".<br><br>Problem: test effort not focused optimally | We took a more disciplined way of identifying in detail the environments needed for the testing efforts. Both the physical and virtual test environments and the production environment were analyzed and identified in detail, and the tools and resources required to be available to the testers in these environments were agreed and identified as early on as possible. | Having a thorough understanding of the entire test environment enables more efficient test design and planning and help in understanding the testing challenges faced in the project. The process of analyzing and identifying the testing environments must be revisited periodically throughout the life cycle of the service. |
| We focused on the functional testing because it is more straightforward to do and specify.<br><br>Problem: non-functional tests did not get focus early enough | We took up the non-functional acceptance criteria with all the relevant stakeholders. The required response times, throughputs, and resource utilization goals and constraints need were documented and agreed as well as all the failure scenarios and monitoring events. All the issues that we could not find criteria for, were assigned on somebody's responsibility for clarification. | The response times are a concern of the users and should be analyzed in detail to enable good enough user experience. Without proper understanding on the user behavior good criteria on the response times is almost impossible to set. The throughput and resource utilization are in most cases not directly user concerns but business and system concerns, respectively. |
| We performed the testing with ad-hoc test data without paying enough attention to the form, amount and nature of the true operational data.<br><br>Problem: test results do not reflect the real situation in operative environment | We started to plan and design the test cases in close cooperation with the end-users, including the type and size of the test data into the test specification. All the key scenarios with determined variability, including all the failure scenarios and related redundancy mechanisms, were specified in detail. | The test data must be realistic in form and in size and appropriate metrics to be collected need to be established.<br>Also, the capabilities to simulate the specified variabilities need to be developed. |
| We did not do the configuration of the test environment in a disciplined enough way and/or the documentation was not updated as regularly and accurately as needed.<br><br>Problem: we did not understand some configurational effects in test results | Preparing and configuring the test environment in a disciplined way and per documented process.<br>Furthermore, we should ensure that the test environment is instrumented for resource monitoring as necessary and that the monitoring system itself is working properly. | The configuration of the test environment requires eye for details and discipline. The preparation of the test environment, all the required tools, and resources necessary to execute tests as new features and components become available for test need to be done in a documented and traceable way. |
| We specified the performance tests only later in the development project, not in the beginning. Furthermore, some of the performance tests were specified without thorough analysis of the true performance criteria.<br><br>Problem: energy was wasted by performing tests that did not provide true value | We developed performance tests in accordance with the other test designs and started to follow more closely the execution and monitoring of the tests. The tests, the test data, and results were validated to ensure that we have measured the right things and that the measurements collected are true measurements of real nature. | The validated tests for analysis should be executed only while monitoring the test and the test environment. In case we observe any exceptional issues these should be investigated thoroughly to understand the root causes of the observations. |
| We did not analyze the test results thoroughly due to some constraints. In addition, the customer was provided the results in a format not easily understandable to all the relevant stakeholders.<br><br>Problem: test results were not effectively used for improving the service | We consolidated and shared the test results to all relevant stakeholders and organized sessions for analyzing the results together with the customer, aiming for better understanding of and new points-of-view to the results obtained. | The analysis and reporting of the test results is naturally the most crucial action of all. A test has been successful only if all the metric values are within accepted limits, none of the set thresholds have been violated, and all the desired information has been collected. |

Based on our experiences, the approach to non-functional testing needs to be well planned in due time before the mission critical system development starts in full. Naturally the testing effort does not stop when the above actions have been performed successfully. Once the testing has been finished with analysis we should remember that we only have tested one scenario on one configuration, nothing else.

Unlike their functional counterparts, non-functional tests should not be developed further all the time, even though we may find room for improvement. Changing the tests hides the real system characteristics, and without a comprehensive understanding on the effect of a certain change, we will gradually lose the control on the system. Naturally, the tests need to be modified and enhanced immediately if there is a real change in the non-functional requirements – in which case we need to admit that some part of our understanding of the system behavior becomes invalid. In our case, we soon also understood the risk that there might exist additional performance acceptance criteria that may not be captured by resource utilization goals and constraints. Thus, we should be able to understand with the help of performance tests also the service behavior with different combinations of configuration settings and gain the knowledge on the most desirable performance characteristics.

The differences between the production and test environments need to be known in detail. One should be almost paranoid about ensuring that the environments are as we think they are. The assumptions made on the environments need to be verified often enough and all the possible changes in the environment must get logged and notified. One important thing to remember is to trust the non-functional test results once assured that the environment has been steady. We failed several times in listening the weak signals observable in the test results.

### 5.3 Service Virtualization and Testing

While it is widely accepted that thorough testing should be performed already early in the service development process, it is often hard to bring effective testing into reality due to the increasing complexity of system environments. To solve this problem, obviously, some new approaches are needed, that preferably both improve the overall level of testing and increase the efficiency of removing the imminent defects.

Service virtualization enables the service provider to create more efficient testing environments by eliminating several issues typically encountered in testing the service. Especially when dealing with the development and delivery of complex services with multiple dependent components that must be thoroughly tested to understand the system behavior, the virtualization comes to rescue.

With service virtualization, we could do the testing of the whole system, from end to end, already before the real dependent services become available. In our project, the service virtualization was used both to emulate the missing elements as well as to compare the behavior of the real system element with the virtualized optimally behaving element. With service virtualization, the test environments can use virtual services in lieu of the production services, increasing the frequency and quality of integration testing.

## 6. SKILLS TO TAME THE MONSTER

From the issues handled above, it is obvious that the skillset required for managing all the aspects in building successfully a mission critical system as a service in much more than just technical skills as such. In retrospect, we were too busy to put the teams together and did not appreciate enough the fact the people working to keep up the service and at the same time keeping the users of the service happy, must be technically first-class but also have strong interpersonal soft capabilities. This skillset is a must to provide customer service constituting a large part of making the mission critical service one that the users want to use.

Technical skills certainly are important as they can help a developer and tester do their job better. But for a software tester also other skills are essential, including the ability to think, the ability to be inquisitive, the ability to be focused, the will to break the rules and the will to fail.

To accomplish the disciplined tasks involved in the provision of a mission critical service and especially the important testing phase, one needs developers and testers with ambition, and as well as people with energy and who want to make a difference (Noll, 2002), (Skulmoski, 2010). Such persons are hard to find and therefore it takes time to create the teams needed for the development effort. Once assured that we have the right people working on the project, trust them fully and make sure that they have the right tools and no impediments on their paths.

In the bigger picture, the industry the service providers represent needs to collaborate with educational providers to guide and inform what skills people need when moving into a tech workplace. Based on our experiences, a part of the people problem is that students at universities are given somewhat outdated material on the requirements they face at the workplace.

## 7. CONCLUSIONS

Information systems of today are in a level of complexity where one cannot trust on the results of traditional testing efforts to gain a reliable knowledge on the system's characteristics. This fact is demonstrated typically by the large number of reported bugs and issued enhancement requests which, on one hand provide us good feedback from the real users, but on the other are also symptoms of customer dissatisfaction.

With mission critical systems, the information related to some problems the users have with the service cannot be left to be channeled through some ticketing system or alike. Monitoring the service in a way that enables the service provider to know about service problems before the users face them is mandatory. In addition, appropriate swift actions to isolate the problem and fix or circumvent it need to be available and planned.

To create a reliable mission critical service, testing is naturally in the key role. Testing a mission critical system needs to be done in a way where the mission criticality is fully considered. We emphasized the role of the automated end-to-end testing of the system in an environment as close to the operative one as was possible. In our opinion, focusing the testing efforts optimally and scheduling the efforts early enough in the development process, saves us from many problematic situations in the operational environment. To conclude, the main lessons learned, regarding the ability to control and understand the mission critical service behavior, are:

- Masked production data is a must for performance testing.
- The test cases should not be developed further and enhanced all the time. Changing the tests hides the real performance and we cannot collect reliable data on the changing system behavior.
- Release and build performance test cycles need to be fully automated early in the development process.
- Understand in detail the differences between the production and performance test environments.
- Trust the performance test results once assured that the environment has been steady.

Furthermore, probably the most important thing enabling a successful provision of the mission critical service is to find right kind of people to do the job. The people need to be truly committed, interested in the customer, have the will to find a solution no matter what and the will to admit that sometimes failure is an important result, something to learn from and not to repeat.

In our project, we faced a multitude of challenges of which the aspects described above are just a small part of. With mission critical system it all comes down to observing and understanding how the service behaves in real life rather than in an artificial environment. And that is how we can achieve service quality: by measuring user impact, and not just preventing bugs, but responding quickly once found.

## 8. ACKNOWLEDGEMENTS

This paper would not have come together without the keen help of our shepherd, Maria Paasivaara from Aalto University, Helsinki, Finland. Thank you, Maria, for your time and valuable ideas and support!

REFERENCES

Canfora, G. Service-oriented architectures testing: A survey. In *Software Engineering* (pp. 78-105). Springer Berlin Heidelberg, 2009.

Dart, Susan. "Concepts in configuration management systems." *Proc. of the 3rd International workshop on SW configuration management*. ACM, 1991.

Dubey, A., & Wagle, D. "Delivering software as a service." *The McKinsey Quarterly*, 6.2007, 2007.

Java Management Extensions, in Wikipedia, Retrieved Feb 28, 2017, from https://en.wikipedia.org/wiki/Java_Management_Extensions

Kumar, Divya, and K. K. Mishra. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market." *Procedia Computer Science* 79 (2016): 8-15.

Landau, Martin. "Redundancy, rationality, and the problem of duplication and overlap." *Public Administration Review* 29.4 (1969): 346-358.

Liu, J. Y. C., & Yuliani, A. R. (2016). Differences Between Clients' and Vendors' Perceptions of IT Outsourcing Risks: Project Partnering as the Mitigation Approach. *Project Management Journal,* 47(1), 45-58.

Lyu, M. *Handbook of Software Reliability Engineering.* IEEE and McGraw-Hill, 1996.

Mission critical, in Wikipedia. Retrieved Feb 20, 2017, from https://en.wikipedia.org/wiki/Mission_critical

New national ERC information system (Feb 28, 2017), retrieved from http://www.112.fi/en/the_erc_reform/new_information_system

Noll, C. L., & Wilkins, M. (2002)." Critical skills of IS professionals: A model for curriculum development." *Journal of information technology education*, 1(3), 143-154.

Paré, G. and J. J. Elam, "Using case study research to build theories of IT implementation," in *Proceedings of the IFIP TC8 WG 8.2 International Conference on Information Systems and Qualitative Research*, pp. 542–568, Chapman & Hall, Philadelphia, Pa, USA, May-June 1997.

Skulmoski, Gregory J., and Francis T. Hartman. "Information systems project manager soft competencies: A project-phase investigation." *Project Management Journal* 41.1 (2010): 61-80.

Stobie, K. "Too much automation or not enough? When to automate testing." *Pacific Northwest Software Quality Conference,* 2009.

Turner, M., Budgen, D., & Brereton, P. (2003). Turning software into a service. Computer, 36(10), 38-44