System Integration Testing in Large Scale Agile: dealing with challenges and pitfalls

KRISTIAN BJERKE-GULSTUEN, ACCENTURE
DANIELA SOARES CRUZES, SINTEF

Imagine driving your car to work every day. During a normal day you pass several toll gates, nothing out of the ordinary on roads in Norway. Now imagine that a couple of weeks later when you check your invoices from these toll gates, you notice that you were charged double the usual cost. The Autosys platform is a complex system of systems (SoS) that forms the foundation for all transactions involving motor vehicles in Norway. The end-user experience consists of value-chain-functionality that integrates more than twenty systems developed and operated by different government departments and agencies. System integration testing is of high importance. However, the project experienced challenges regarding unaligned SoS development plans, uncomplete SoS integration designs and SoS integration test scenarios and insufficient test automation coverage. In this experience report we describe how the Autosys project dealt with these challenges.

1. INTRODUCTION

Modern software applications and systems are rapidly evolving to become more sophisticated with complex multi-directional dependencies to several other integrated systems. The components, once developed, are integrated and ultimately tested in the deployment environment. System Integration (SI) is concerned with forming a coherent whole from component subsystems, including humans, to create a mission capability that satisfies the needs of various stakeholders [1]. There are different forms of SI. Historically, vertical integration occurs when components of a system developed by a single acquisition program are integrated to produce the desired capability. Currently, vertical integration has a broader meaning which covers both a single organization and joint organizations which have multiple acquisition programs over time that contribute to the creation and enhancement of a mission capability. Horizontal integration occurs when systems developed by different acquisition programs, often for different customers and purposes, are brought together to create a new capability. A key goal of SI is to ensure that semantic and syntactic interfaces between component elements of the system perform as specified by "contracts" between the elements. A companion goal is to ensure that the interfaces can be adapted in well-understood ways with relatively modest effort. Almost all SI failures occur at the interface level primarily due to incomplete, inconsistent, or misunderstood specifications. Invariably, the root causes of failure tend to be ad hoc integration and failure to develop and adhere to formally defined semantic concepts and relationships [1].

A system of systems (SoS) is a collection of systems originally designed as stand-alone systems for specific and different purposes but that have been brought together within the SoS umbrella to create a new capability needed for a particular mission [2]. An SoS may be formed dynamically to perform a given mission and then reorganized as needed for other missions including those that have not yet been envisioned. A good SoS design might have modules that are not as good as their stand-alone counterparts that perform the same functions. System of Systems integration (SoSI) takes on a new meaning that comes with a host of new challenges. Once the component systems are developed, they are incrementally integrated and tested and, ultimately, deployed in the operational environment.

In this experience report we describe SoS integration testing challenges and pitfalls as experienced by the Autosys project. The Autosys platform is a horizontal SoS where end-user experience consists of value-chain-functionality that integrates more than twenty systems developed and operated by different private and public departments and agencies. According to the Systems Engineering Guide for SoS [3,4], an SoS can be classified according to the way it is managed and its openness to change and new capabilities. The form and rigor of SoS is directly related to SoS type. The Autosys platform can be categorized as an *Acknowledged SoS*: having recognized objectives, a designated manager and resources. Its constituent systems retain independent

Author's address: Kristian Bjerke-Gulstuen, Rolfsbuktveien 2, 1364 Fornebu, Norway; email: k.bjerke-gulstuen@accenture.com
Second author's address: Daniela Soares Cruzes, SINTEF, NO-7465 Trondheim, Norway; email: daniela.s.cruzes@sintef.no
Copyright 2020 is held by the author(s).

ownership, objectives funding, development and sustainment. Changes in the system are based on collaboration between the SoS and the system. The Autosys project experienced delayed availability of complete and testable value chains (E2E-testing), resulting in late identification of defects and unidentified dependencies. This caused a growing lack of trust in the new system and stressed the project as the release deadline approached. Important success factors were to support the Scrum teams by establishing a dedicated SoS integration team responsible for the follow up on dependencies and delivery plans from all systems in the SoS, early involvement of testers with extensive use of the testers mindset during integration design and to increase our focus on automated verification of data used in the different value chains.

Kristian Bjerke-Gulstuen is an experienced test manager in Accenture. He has been responsible for planning and executing testing in several large-scale projects, using both traditional and agile development methods. In the Autosys Project Kristian is the test manager from Accenture, responsible for defining the project's test processes and the management, planning and execution of all quality engineering and test activities performed by Accenture. Kristian is part of the project management team and has been with the project since 2016. His experience from the project has been discussed with researcher Daniela Soares Cruzes from SINTEF who has a focus on testing, quality assurance and agile development.

The remainder of this report is organized as follows: Chapter 2 describes the project context and background. Chapter 3 describes the main SoS integration testing challenges and how the project collectively dealt with the challenges. Chapter 4 summarizes the key learning points and provides recommendations to other projects doing SoS integration testing.

2. BACKGROUND

The Autosys Project is a large-scale agile digitalization project that is managed by The Norwegian Public Roads Administration (NPRA). NPRA is a Norwegian government agency responsible for the construction and maintenance of highways and county roads, including the supervision and administration of registered vehicles and certifications. The project's main objective is the digitalization of business processes related to vehicles and to replace the legacy Autosys automotive register with the new Autosys vehicle application platform. The Autosys platform is considered critical to the Norwegian society as it operates the formal approval, by law, of vehicles, registration and change of ownership, reseller solutions, and distribution of data to other public administrations and selected private partners. Autosys is an SoS where end-user experience consists of value-chain-functionality that spans several systems that are developed and operated by different government departments and agencies (such as the police, Norwegian tax authorities, insurance companies and road toll companies). The new platform is managing 11 million vehicles and is the core of an immensely complex ecosystem with an annual turnover of 300 billion NOK.

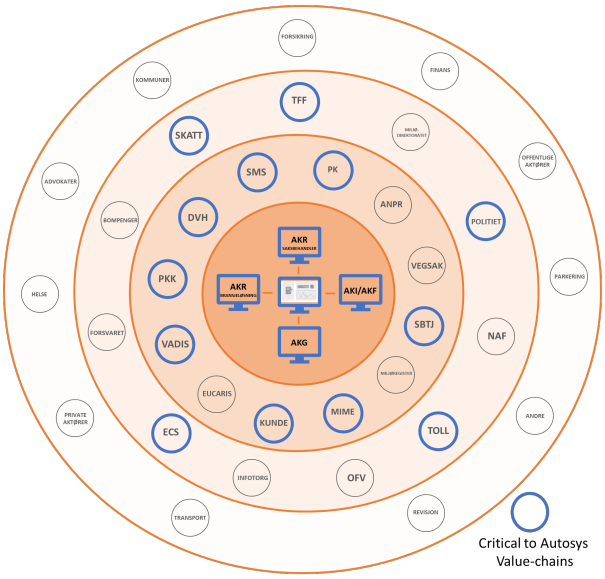

Figure 1. Illustration of the Autosys platform as a horizontal SoS

When developing the new Autosys platform, modifications to more than twenty of the existing integrated systems were required. These changes were necessary as the integration technology and architecture would change in the new platform. Figure 1 illustrates Autosys and its horizontal system integration landscape. In order to complete the many functional value-chains, Autosys distributes transaction data related to vehicles to several systems. Relevant changes in data are received by several internal and external systems, transactions are processed and, in many situations, fed back to Autosys for the completion of a business process. Changes in data may be consecutively reporting on updates in the vehicle register, i.e. change of ownership or temporary deregistration, emission data and much more. Other integrations can be data look-up (single and bulk) of vehicles to provide information on selected technical data at a given time, i.e. additional information such as historical data (ownership, number plates, and registration statuses).

The Autosys project planning phase started in the summer of 2014 and the project was initiated together with Accenture as the selected IT-partner in August 2016. Accenture is responsible for development of the core part of the system (the inner circle in Figure 1). Approximately 120 people are daily involved in the development process of the solutions. The project is scheduled to be completed in 2021.

2.1 Autosys delivery model and approach to agile development and testing

The project uses a PRINCE2-based model and the development phase uses a hybrid agile/Scrum development model (based on PS2000 SOL). The following Scrum practices are used in full or partial by the project: unit and integration testing, continuous integration (CI) and continuous testing, weekly and daily stand up meetings, incremental design, daily deployment, test driven development, test automation, customer side involvement, planning poker, negotiated scope contract, retrospective, epics and user stories, fixed cycles and team continuity. Epics and user stories were defined by NPRA. Acceptance criteria were specified at user story level during the initial and continuous solution design phase. If necessary, new user stories and acceptance criteria could be added during the sprints.

A "big bang" transition to the new system was considered undesirable, hence the delivery plan prepared for agile development with a staged phase-out of the legacy system. The planned replacement process included seven main deliverables and two main deliverables deployed to production per year. Maintenance releases are developed in parallel and deployed when needed. The development phase for a main release consisted of 7-12 sprints, of three weeks each.

Quality engineering and testing had a high priority in the project, and with extensive focus on "shift-left testing". The following quality engineering practices were incorporated in the different project phases: automated and continuous testing, manual and structured testing, exploratory testing, static analysis and static testing, test driven development, experimentation, learning and demos. Non-functional testing such as security, performance and operations testing was done both during the sprints and as part of the system- and user acceptance (UAT) testing. Our process for quality engineering and testing evolved constantly, however the basics in terms of test phases, test activities and test organization has been consistent and as illustrated in Figure 2.

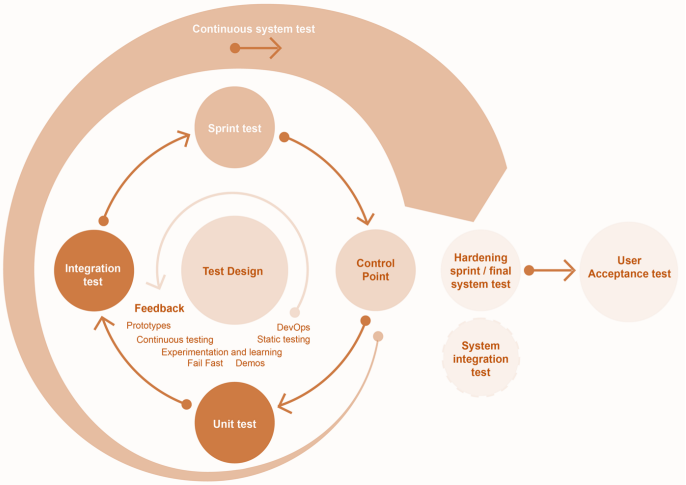

Figure 2. Test process at the Autosys project

The IEEE defines integration testing as testing in which software components are combined and tested to evaluate the interaction between them [5]. Integration testing was an important part of our testing process. At the Autosys project integration testing consisted of low level and high level integration testing. The low level integration testing focused at the interface level and was performed by the developers during all the sprints. High level integration testing was done during system test and included SoS integration testing. All low level integration tests covered normal cases and special cases; they were developed in jUnit, incorporated in the CI pipeline and reported in Sonar. Low level integration testing relied heavily on stubs and mocks. High level integration testing focused on SoS integration testing preferably using production candidates of all systems in the SoS. The scope of high level integration testing was testing of user scenarios that spanned more than one system in the SoS. This testing was performed in parallel with the sprints and should start as soon as all necessary user stories were deployed to the test environments. The testing was both manual and automated.

Test design and test scenarios per user story and at system/SoS-level were developed as part of the continuous solution design phase. Unit test, integration test (using stubs) and sprint test were done as in-sprint test activities per user story. After each sprint NPRA was responsible for executing a control point. The control point consisted of a demonstration of the sprint deliverable (sprint demo) and a verification phase to make sure all deliverables were completed according to the Definition of Done. Continuous automated and manual testing was performed in parallel with the sprints. The sprints were followed by a three-week final system test (hardening sprint), and a 6-8-week user acceptance test performed by NPRA.

3. DEALING WITH SYSTEM INTEGRATION PITFALLS AND CHALLENGES

Continuous design and development kept a steady pace, however when transitioning to high-level integration testing, we started to experience several challenges:

- Unaligned SoS development plans
- Uncomplete SoS integration designs and SoS integration test scenarios
- Insufficient test automation coverage

3.1 Dealing with unaligned SoS development plans

Challenge. When starting high level SoS integration testing we experienced that insufficient coordination of SoS deliverables caused unaligned development and test plans, hence delays to the planned SoS integration test execution. The scope to be implemented, dependencies and timelines important to the Autosys project was communicated and discussed with all system owners in the SoS at the start of every release. However, the regular follow up was done in a fragmented and ad hoc fashion. This soon escalated to unsynchronized plans and delayed the availability of production candidates of all integrating systems in our test environments. The test team experienced different scenarios, as illustrated in **Error! Reference source not found.**. I.e. according to the development plan the SoS integration test team could start SoS integration testing in sprint 4, however several of the integrating systems would in reality not be ready until UAT. It became challenging to accomplish the goal of executing an early and structured SoS integration test of business processes

Our designers, developers and testers found it challenging to have a complete overview and control of how transaction patterns and data combinations could affect all other systems in the SoS. This was a known risk and challenge, and our initial and preferred mitigation was to identify defects through early SoS integration testing. However, having delays to the SoS integration test execution resulted in increased risk of identifying integration defects and misunderstandings late.

Another issue was that other projects and owners of the other systems in the SoS used different Jira instances than Autosys, and we had no tool that supported linking between the dependent user stories. Due to these challenges the quality of the solution was questioned by several system owners and a dawning lack of trust in the project's capability in meeting the deadlines pushed the project management to act.

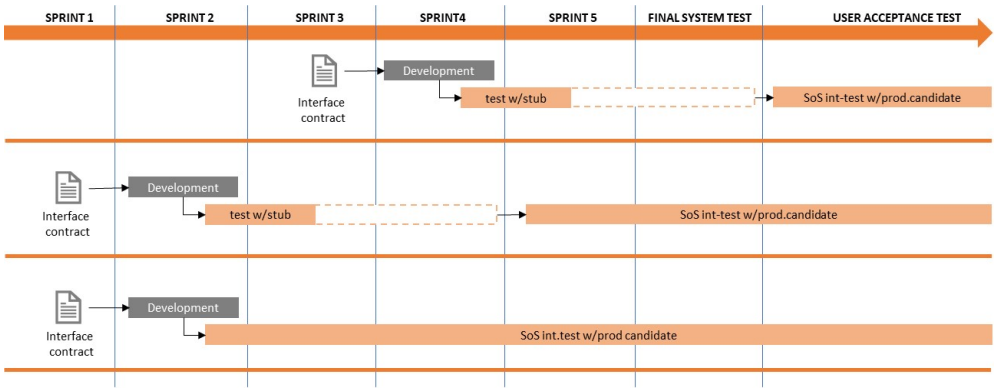

Figure 3. Delays resulted in late SoS integration testing with production candidates of all systems

Our solution. Due to the complexity and scope of the SoS integrations we learned that the design and Scrum teams needed support in dealing with SoS integrations. The project management decided to strengthen the SoS integration team to gain better control with SoS integration deliverables, and with extensive focus on the systems that were not within direct control of the Autosys project. Our priority was to make sure deliverables that had dependencies was delivered in a timely fashion, reducing the use of stubs in high-level integration testing. The SoS integration team had extra personnel added to the team; and representatives from the Scrum teams, test managers and selected system owners from the integrating systems participated in weekly stand-up meetings. The technical integration and interface specifications were rarely an issue; however, the development of corresponding modifications could have delays. These meetings therefore focused on sharing information on status and discussing potential actions and work arounds if delays were reported.

To support the status and progress reporting we modeled all dependencies using Jira. The SoS integration team got access to all relevant Jira instances, and user stories with dependencies to other systems were tagged in Jira using a custom field. The SoS integration team decided to use Jira-linking to group all user stories with dependencies. The concept is illustrated in Figure 4. The team then used different link types to support the tracking of the status of user stories necessary to be completed before SoS integration testing without stubs could be performed. Due dates on the user stories were set on all relevant user stories and aligned with the test plans. By using this method all dependencies were automatically displayed in Jira. The flowing link types were used:

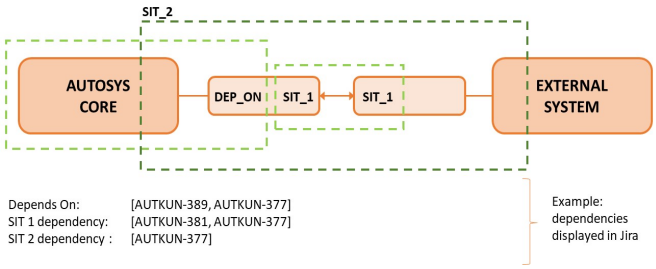

Figure 4. Jira link types documenting user story dependencies

- DEP_ON (dependent on): user stories that were required to be delivered in front of Scrum team development. I.e. interface contracts.
- SIT_1 (system integration level 1): user stories that needed to be implemented to enable connection to the external systems test environment, including access to stubs. These user stories had to be delivered to start first wave of high level SoS integration testing.
- SIT_2 (system integration level 2): user stories that needed to be delivered before complete functional business process testing could be executed with production candidate versions of the external systems.

The goal was to complete integration testing at SIT_1-level by completion of the final system test, and that all necessary user stories linked with SIT_2 were released to the test environment upon UAT start.

The SoS integration team lead and SoS integration test manager collaborated on the status reporting and reported weekly to the project management team. As the test team relied heavily on risk based testing, a new status reporting template combining status of the SoS integration testing and the test effort was established (Figure 5). The report also provided information on whether SoS integration testing using production candidates or stubs had started.

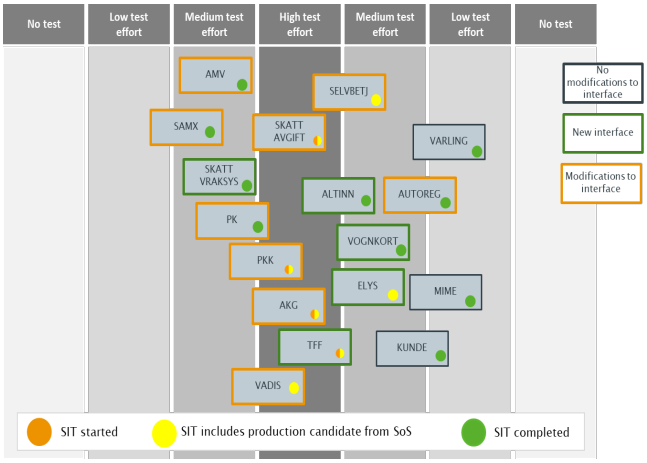

Figure 5. Combining SIT status and test effort reporting

Results. Having a dedicated team continually doing structured coordination and follow up on delivery plans, dependencies and progress, the project lowered the risk of not meeting the deadlines due to SoS integration testing delays. By the start of the UAT we managed to have most of the SoS value-chain tested at SIT_1-level and SIT_2-level integration testing had started for several of the systems. The SoS coordination and follow up continued through all sprints and the team continued their work during all following releases. We did not experience having an SoS integration component team as defragmentation of responsibilities or as any negative to the projects or the agile ways of working.

Establishing the integration dependency model in Jira was done during two sprints but needed frequently reviews to verify if newly added user stories should be linked. Dependencies that were displayed automatically in Jira gave the teams a complete picture of the SoS integrations, including statuses, and it became easier for the SoS integration test team to create their testing plans and identify potential delays.

The new status report was used on a weekly basis and was found understandable and easy to communicate when reporting status and progress to the project management and steering committee. Frequent reporting involved the project management in the details of SoS integrations and integration testing complexity, resulting in better understanding of the challenges.

3.2 Dealing with uncomplete SoS integration designs and SoS integration test scenarios

Challenge. Even though we improved at having user stories delivered in sequence aligned with the test plans, hence fewer delays in starting the testing, we experienced that defects, misunderstandings and even conflicting requirements blocked the SoS integration testing. We experienced that we identified more new user stories than expected related to SoS integrations when doing final system test and UAT, and realized that it was extremely difficult to have complete insight and understanding of how data was used across all systems in an SoS legacy system that had evolved trough 40 years. This made it challenging to design SoS integration test scenarios that would cover all necessary paths through the systems, increasing the risk of not identifying blockers and defects. This again put a high pressure on the project as the time to fix these issues became shorter.

Our solution. The SoS integration test team analyzed the situation together with the SoS integration designers and they decided to focus on improving both the integration designs and the SoS integration test scenarios. The two teams approached this task by introducing a concept we named Test Driven Design. The basic and simple idea behind this concept was to gather resources from different SoS stakeholders, including system owners, designers, developers and SoS integration testers, and "think as a tester" when discussing and experimenting with the different functional and technical scenarios involving SoS integrations. We named it Test Driven Design rather than Test Driven Development to stress the fact that the improvements were in the integration designs and test scenarios, and not at the development.

The activity was done having the resources collocated and it was done in iterations. The method was structured walk-throughs on paper and by using both the deployed new version of the Autosys platform and the legacy system still running in production comparing different transaction sequences. The priority was to get a unified understanding on how the planned modifications would affect user scenarios spanning the SoS, and the level of criticality and consequences to the end users if there were defects that would trigger ripple effects such as faulty calculations in integrated systems.

If we identified delays to the development plans for single systems in the SoS that could not be easily adjusted, we still started SoS integration testing by doing static testing activities. The static testing was mostly done as reviews and walk-throughs using the design documentation and the learning from the test driven design activities as test basis instead of the deployed code.

Results. The creative minds of the tester and their "what if …" mindset was experienced as very helpful when improving the completeness of the integration designs. A better understanding of transaction sequences and how data was altered during the steps in a business process was very useful to the SoS integration testers when developing the test scenarios. We learned that even minor variations in specific data fields in the Autosys core systems could have a huge impact on calculations further down the value chains.

We also learned that most "happy cases" usually worked fine and as designed. It was when we decided to broaden the test scenarios with combinations of user scenarios not in the "happy case" sphere, we found the more complex defects. I.e., doing a relatively straightforward change of ownership produced the expected result in the integrating systems. However, when doing a change of ownership and then corrections to certain data fields registered for the vehicle, the altered data fields were not mapped correctly resulting in too high or low calculations of taxes and insurance fees (in two of the integrated systems).

Doing static SoS integration testing when we had delays to single systems or isolated functionality in the integrating systems was found to be a very valuable activity. Even though we couldn't actually see the actual results produced by the systems we could still do high level analysis and evaluation of the completeness and correctness of the user stories implemented. By constantly having the test teams push and think "shift-left" and "nothing is going to block our testing", also motivated the other teams to constantly strive for improvement.

The project continued to use the test driven design technique upfront of SoS integration test execution as it really helped us to early validate if all requirements and acceptance criteria were met. The upfront effort resulted in more efficient SoS integration testing and a reduced number of integration defects during UAT. We experienced that more and more new user stories were identified and implemented during the sprints rather than during the UAT. All this resulted in a higher confidence among the SoS stakeholders and that the project would be able to deliver complete and correct SoS integrations, hence meet the agreed go-live dates with a high quality solution.

3.3 Dealing with insufficient test automation coverage

Challenge. The Scrum and test team had implemented a thoughtful approach to test automation, including test automation as part of CI and automated value-chain testing as data driven automated tests. Our strategy was to

develop relatively few tests focusing on core functionality that could test larger data sets rather than many smaller testes covering the entire solution. Our automated tests did a good job in testing the Autosys core systems isolated, however they did not cover larger volumes of data combinations and user scenarios doing modifications to data important to calculations and transactions in integrated systems. Hence important defects were missed. From the Test Driven Design, we saw our automated tests mostly covering "happy cases". The automated tests did not uncover the defects that we identified during manual SoS integration testing, as these test scenarios now covered more complex transaction patterns.

Our solution. When phasing out a legacy system we had the luxury of having a blueprint for how data were expected to look like to the other systems in the SoS. By replicating vehicle transactions from the production system to the test environment and comparing the result, we were able to implement automated checking routines that could run on thousands of vehicles. Given that we had a constantly changing new core system in development, there was a need for quick verifications to uncover if unforeseen changes had occurred in our systems. Based on the experiences from our Test Driven Design we embraced this opportunity and developed automated checking programs, in Java and Excel, that replicated vehicle changes from production in a test environment and then compared the result to the vehicle in the two environments. This replicate-and-compare job was set up to run regularly via Jenkins, ensuring that we were notified quickly if an unforeseen change occurred. The concept was fairly simple; however, it was an extremely valuable addition to our testing. For change of vehicle ownership, the check implemented would look like this:

1. A change of ownership occurred in production environment;
2. The verification program picks up the change of ownership transaction;
3. Do the same change of ownership in the test environment;
4. Look up the vehicle in the production environment as it is now (after the change of ownership);
5. Look up the vehicle in the test environment;
6. Expect the same response at 4 and 5;
7. Send notification if any deviations.

Results. The Autosys project identified and corrected defects directly after new code was deployed to the test environments by automatically and frequently comparing data regarding several thousand vehicles. The checking covered transaction patterns as done in the production environment, it was done before the manual testing and even better; before other systems in the SoS could identify the defects. Now, the automation strategy and implementation worked better as the safety net it was meant to be.

This activity made us realize the power of checking and how important this is as a supplement to the testing of a system. Automating the checking helped us free up time for the testers to do more exploratory testing and to, through experimentation and learning, give better evaluations on the usability and quality of the solution.

Evolving our test automation strategy also resulted in a positive collaboration between testers and developers. They designed the checks together, and we experienced that the two disciplines were brought even closer together; the testers developing their technical skill set and the developers to be better at understanding the importance of demonstrating that the code will work when thinking outside the pre-defined and designed user scenarios.

4. KEY LEARNING POINTS

The SoS integration testing process was critical to the delivery of the final product in the Autosys Project. However, the efficiency and quality of the SoS testing was affected by several factors. By dealing proactively with the issues and shifting quality assurance and testing activities left instead of adding test phases to the right we regained control of the project progress and solution quality. All go-live dates were met as planned and we experienced satisfied end-users and very few major defects after go-live: the project is considered a huge success. Some SoS integration defects leaked to the production environment and the defect scenario described as part of the abstract actually happened. The defect was quickly fixed; however, it was a huge reminder to all, and our **first key learning point**; even when implementing several actions regarding SoS integrations and testing, you cannot guarantee that no defects will leak to the production environment. You constantly need to stay alert and actively make use of all learning and experiences before, during and after a solution is set into production.

The second key learning point was that an SoS project should support their teams by having an organizational project unit, person or team, responsible for coordinating SoS integrations in scope. This

includes alignment of plans, reporting, communication, dependencies, deployment routines, test data and routines for defect fixing. In addition, the team needs to manage that user stories and defect fixing are requested and delivered according to the sprint plans enabling early testing with production candidates in the test environments. The project unit must be supported with a tool for tracking status and progress across different projects/organizations/state departments/state authorities. When the Autosys project is completed and goes into "operations mode", we plan to keep the component team operational, as part of portfolio management, when developing larger releases and gradually pass the responsibility to the Scrum teams when developing features.

The third key learning point was that the greater value from system integration testing arrives when doing testing without any use of stubs and mocks. Many SI failures occur at interface level due to incomplete, inconsistent or misunderstood specifications. However, to us, lack of knowledge and insight regarding the integrated systems and their use of Autosys data was an important additional source of SI failures. It was very challenging to have control of how all transaction patterns and data combinations could affect other systems in the SoS, hence early testing with the production candidate was the best way of identifying defects and misunderstandings. It was during this testing we were able to really challenge the product and identify the larger integration issues. Furthermore, SoS integration testing should be done by subject matter experts representing the different systems and in combination with skilled test experts. If dynamic testing without using stubs is not possible until late sprints, test driven design involving relevant project teams across organizations should be performed during the design phase.

The fourth key learning point was to continuously (and automated) check that new code does not introduce defects. The automated checking must run on volumes of data to demonstrate that as many as possible combinations of data produce the same result in the old and new solution. The automated checking should be performed as a supplement to the manual testing and other automated testing. The automated checking is important for identifying regression defects and is a good source to identify misunderstood requirements. The mechanism is important as it frees up time for the testers to focus on experimentation and exploratory testing.

The fifth key learning point was that the team doing business process testing of the SoS (not only interface testing/technical integration) needs to be staffed with SoS integration testing experts and functional domain experts. It is important that all test team members understand the complete value-chains in the SoS, including dependencies and the consequence of potential defects leaking to production for all the integrating systems.

The sixth key learning point was that system integration projects such as Autosys should absolutely be delivered in an agile way, however the level of agility needs to be discussed and agreed upon when starting the work on a project release. If the level of insecurity related to data usage cross systems and value-chains are medium to high, extra measures in terms of additional planning, follow up on definitions of done etc., should be added to the daily follow up and management routines.

5. ACKNOWLEDGEMENTS

We are grateful to Piet Syhler for helpful comments and guidance on this experience report. We thank the NPRA and Accenture for all support during the process of writing this report. This work was supported by the SoS-Agile (247678) project, funded by the Research Council of Norway.

REFERENCES

[1] A.M. Madni and M. Sievers, Systems integration: Key perspectives, experiences, and challenges, INCOSE J Syst Eng 16(4) 2013.
[2] W.H.J. Manthorpe, Jr., The emerging joint system of systems: A systems engineering challenge and opportunity for APL, Johns Hopkins APL Tech Dig 17 (1996), 305–313.
[3] Madni, Azad M., and Michael Sievers. "System of Systems Integration: Fundamental Concepts, Challenges and Opportunities." Advances in Systems Engineering (2014): 1-34.
[4] Office of the Undersecretary of Defense for Acquisition, Technology, and Logistics (OUSD AT&L), Systems engineering guide for systems of systems, Washington, DC, August 2008.
[5] Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology: ANSI/IEEE Standard 610-12-1990. IEEE Press: New York, 1990.