Testing Highly Complex System of Systems: An Industrial Case Study

Nauman Bin Ali
School of Computing
Blekinge Institute of Technology
37140 Karlskrona, Sweden
nauman.ali@bth.se

Kai Petersen
School of Computing
Blekinge Institute of Technology
37140 Karlskrona, Sweden
kai.petersen@bth.se

Mika V. Mäntylä
Department of Computer Science
Lund University
22100 Lund, Sweden
mika.mantyla@cs.lth.se

ABSTRACT

Context: Systems of systems (SoS) are highly complex and are integrated on multiple levels (unit, component, system, system of systems). Many of the characteristics of SoS (such as operational and managerial independence, integration of system into system of systems, SoS comprised of complex systems) make their development and testing challenging.

Contribution: This paper provides an understanding of SoS testing in large-scale industry settings with respect to challenges and how to address them.

Method: The research method used is case study research. As data collection methods we used interviews, documentation, and fault slippage data.

Results: We identified challenges related to SoS with respect to fault slippage, test turn-around time, and test maintainability. We also classified the testing challenges to general testing challenges, challenges amplified by SoS, and challenges that are SoS specific. Interestingly, the interviewees agreed on the challenges, even though we sampled them with diversity in mind, which meant that the number of interviews conducted was sufficient to answer our research questions. We also identified solution proposals to the challenges that were categorized under four classes of developer quality assurance, function test, testing in all levels, and requirements engineering and communication.

Conclusion: We conclude that although over half of the challenges we identified can be categorized as general testing challenges still SoS systems have their unique and amplified challenges stemming from SoS characteristics. Furthermore, it was found that interviews and fault slippage data indicated that different areas in the software process should be improved, which indicates that using only one of these methods would have led to an incomplete picture of the challenges in the case company.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Diagnostics

ESEM'12, September 19–20, 2012, Lund, Sweden.

General Terms

Practice

Keywords

System of Systems, Software Test, Case Study

1. INTRODUCTION

System of systems (SoS) recently received vast attention in the software engineering research literature. A system of systems is characterized through operational and managerial independence in the development of the individual systems that should later on act together, and is characterized by an integration of many different systems into a new system. System of systems are also generally very complex, and there exist suppliers that deliver them for integration [16, 7]. Literature (cf. [19, 8]) distinguishes different types of SoS, namely virtual, collaborative, acknowledged, and directed SoS.

It is acknowledged that SoS development and quality assurance is very challenging, e.g. due to involvement of many parties, it is not easy to integrate systems continuously, and so forth [19, 16, 17, 7]. However, so far there is a lack of empirical studies that explore the challenges and possible solutions of how to test such complex system of systems. In response to this research gap this case study makes the contribution to investigate the challenges and potential solutions of SoS development in an industrial case study of a large-scale system of systems from the telecommunication domain with over 5,000,000 Lines of code. The following contributions are made by this study:

- Understand how SoS testing is done by describing and characterizing how system of systems are currently tested based on a case having the typical characteristics of directed SoS development.
- Identify the challenges of SoS testing observed in the case and categorize them to three classes: a) challenges that are not different in SoS context and other contexts, b) challenges that are amplified in SoS context but that can also be found in other contexts, and c) new challenges specific only to SoS context.
- Identify possible solutions of SoS testing based on the challenges.

The remainder of the paper is structured as follows: Section 2 presents the related work. Section 3 describes the

research method, followed by the results in Section 4. Section 5 concludes the papers by presenting the observations and implications given by the results.

2. RELATED WORK

The Systems Engineering Guide [11] defines SoS *"set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities"*. Maier [19] and Dahman et al. [8] classify SoS into Virtual, Collaborative, Acknowledged, and Directed. Virtual means that there is no central management or agreed purpose of the SoS, purposes emerge as systems are combined. This reflects the philosophy of service-oriented architecture, which is a way of implementing SoS [17]. Collaborative SoS have an agreed purpose and interact voluntarily. How systems should interact is collectively decided between system owners. Acknowledged SoS are characterized by common objectives, and there exists a designated manager that assures resources for SoS development. The System Engineering Guide stresses that the individual systems still have independent ownership, objectives, funding, and development. Directed SoS are built to fulfil specific purposes and they are managed around those purposes. There exist a central management for the SoS. However, the system development organizations contributing individual systems are still independent, but subordinate to the purpose. The SoS in this study is characterized as directed.

The System Engineering Guide [11] also highlights some engineering challenges that are particular to SoS, namely (1) Management issues with respect to governance of system development across organizational boundaries, (2) the increased complexity, scope, and cost of processes for planning and engineering, (3) achieving and maintaining interoperability, and (4) likelihood of unpredicted behaviour, and hence the lack of predictability of behaviour.

Lewis et al. [16] point out that SoS challenges are related to collaboration, in particular who collaborates and what everyone has to provide so that the collaboration would work. Furthermore, there are particular challenges if systems do not evolve in a similar pace or manner, which makes their integration challenging. Hence, if there are some dependencies (one system cannot fulfil its service without another) then there is a need for synchronization, which makes collaboration essential. How the SoS is built in an enterprise affects the severity of the challenge of collaboration. They distinguish SoS in an enterprise where a single or multiple organizations develop, and where multiple enterprises develop.

Columbi et al. [7] characterizes SoS as systems that operate synergistically, however, SoS can operate and be managed independently. They acknowledge and highlight that these characteristics make testing and evolution of SoS very challenging. In other words, they have amplifying character, or even lead to completely novel challenges. In their experience report from a SoS development at the Department of Defense (DoD), they suggest a number of challenges: (1) organizational structures do not support testing in a SoS framework; (2) No steps in testing that evaluate the overall SoS capabilities, rather the focus is very much on individual systems (no obligation for the project manager to scope testing on the SoS level), (3) Rate of incoming requirements has drastically increased on the SoS, leading to an exponential increase of test events, complexity, and expense with respect to testing; (4) No overarching test scenarios across different systems due to coordination and communication breakdowns. In response to the challenges, they provided a number of solutions: (1) only test interoperability with respect to changes made, and do not test everything, but rather *"built a little, test a little"*, (2) use risk assessment to prioritize the ever growing number of tests, (3) focus tests on interfaces as there the greatest risk of mistakes lie, but do not neglect internal behaviour of systems, (4) provide designated environments and people for integration.

Furthermore, we identified other studies that have a focus on SoS quality assurance (see e.g. [13, 5]), but those are solution proposals without an industrial partner involved.

Overall, we found that there is a lack of systematic empirical evidence that investigates how to conduct testing in a SoS context, and relates that to challenges and solutions.

3. RESEARCH METHOD

As a research method we used case study research, following the guidelines provided in Yin [24] as well as Runeson and Höst [22].

3.1 Purpose and Research Questions

The purpose of the study is to gain an in-depth understanding of testing practices, challenges, and potential solutions when testing very large SoS.

RQ1: How is SoS testing done in industry? We first need to gain a deep understanding and rich description of the current situation, as SoS testing is not well described in literature. The case being studied in that sense provides an interesting case as it fulfils the characteristics of directed SoS development very well.

RQ2: What are perceived and measured challenges when testing SoS and how are they different from testing challenges of other contexts? Given testing in SoS is not well explored in industry the first step should be to understand the challenges in order to find useful solutions.

RQ3: What potential solutions do practitioners see in order to address the challenges identified? The identification of the solutions is based on experiences made in the SoS context. They provide useful input for future evaluations to test their actual impact in the studied context.

3.2 Case and Context

The case being studied is a development site of a large Telecom vendor, engaging in SoS development. The case and context are described, as this allows for generalizing the results to a specific context. Other companies in a similar context are likely to find the results transferable to their context [20].

The process used at the company follows a SoS approach. There is no common definition of system of systems, as the term has been defined in different domains, such as military, enterprise information systems, or education [15]. The term has been recently established in the software engineering field, where a system of systems should fulfil several of the characteristics. These characteristics and their presence in the case company are shown in Table 1.

The overall architecture of the system of systems studies consists of 12 systems, which are operationally independent and can also provide services independently of each other. The process used at the company is shown in Figure 2. In

Table 1: System of Systems Approach Characteristics (cf. [15])

ID.	Characteristic	Case Company
C1	Operational independence	(√)
C2	Managerial independence	(√)
C3	Integration of system into system of systems	(√)
C4	SoS comprised of complex systems	(√)
C5	System suppliers deliver systems for integration	(√)
C6	Complete technical overview of SoS and system supply	(√)

the first step the high-level requirements for the overall SoS are specified. Before the requirements are handed over to compound system development a so-called "Go"-decision is taken, meaning that development resources are allocated to the high-level requirement. When the decision is positive, teams specify a detailed requirements specification, which then is handed over to the concerned system(s). The requirements are then implemented for a specific development system, and they are integrated (also called system level test).The development is done in sprints run by agile development teams (AT Sprints in Figure 2). Each system can be integrated independently of another system, which provides them some degree of operational and managerial independence (see Table 1). However, the versions of two systems have to be compatible when the system of systems is integrated (Compound System Test). Each of the systems is highly complex, the largest system having more than 15 development teams. The size of the overall system of systems measured in lines of code (LOC) is 5,000,000 LOC. This fulfils the characteristics of SoS development related to system complexity and integration. In order to make sure that the system of systems is working together in the end an overall system structure and design is developed, referred to as the anatomy. This allows having an overview of the overall SoS, also making explicit how each system in the SoS contributes to the overall system goals.

Looking at other context elements [20] the following should be added as information:

- All systems and the SoS are older than 5 years.
- On principle level the development process is incremental with projects adding increments (e.g. new functionality) to the code base-line on system and compound system level.
- Within the teams and in the testing activities agile practices are used, such as: continuous integration, time-boxing with sprints, face-to-face interaction (stand-up meetings, co-located teams), requirements prioritization with product backlogs, re-factoring and system improvements

3.3 Data Collection

We used multiple data sources for triangulation to increase the validity of the results. In order to answer RQ1 and RQ2 we used documentation, interviews as well as quantitative data as input. For RQ3 we used interviews to gather opinions of how to best improve SoS testing.

Process documentation of the test process is obtained in order to capture the specified test strategies and test levels conducted at the company. This information is used for triangulation purposes as an additional source to the interviews. Furthermore, the documentation aided the researchers to gain better domain knowledge and familiarity with the test specific terminology used at the company before conducting the interviews. In addition we consulted official templates for reporting (e.g. forms for documenting defect data).

Interviews It is important to point out that the sampling procedure here is not to get a representative sample of a population, but rather having a diversity of perspectives to be able to contrast alternatives in the evaluation step. We notice that with each additional interview there was a high overlap in the views and we decided to stop after five interviews. Even though the practitioners had diverse roles the number of challenges and improvement suggestions stabilised after the third interview, which indicates a good coverage of the interviews (refer to Figure 1).

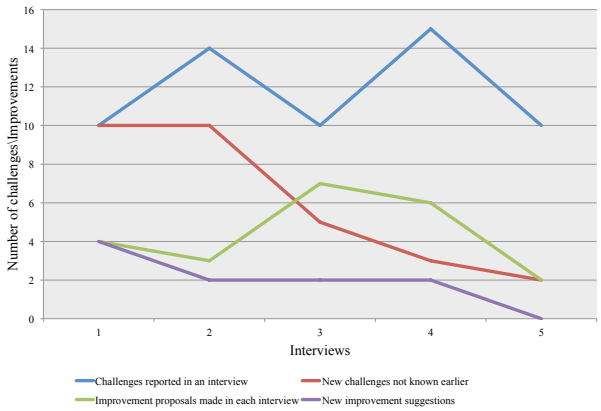

Figure 1: Stability in challenges/improvement suggestions identified in interviews

Test Managers who have the knowledge about their team members supported us in interviewee selection ensuring diversity and coverage of various test levels in the company. In the following a short profile of the testers is provided in Table 2.

Table 2: Practitioners' profile

ID	Description
1	The interviewee is currently working with test improvement with a particular focus on FT and ST, and has experience from working in testing teams before. Furthermore, the interviewee has more than five years of experience focused on testing.
2	The interviewee is working in software development since 1995 and is currently a technical coordinator. In the past the interviewee also worked as a tester, and had a leading role in design teams with responsibility for product and test code.
3	The interviewee is a technical expert assisting software developers in solving technical problems, in particular the problems that occurred in testing. Hence, the interviewee is directly exposed to review and BT in the daily work.
4	The interviewee is working with the company since 1994, and has been involved in software testing, development, support, and system management. She has worked as a developer for several systems in the company, currently leading a development team investigating new enhancements to the main system. The interviewee has experience of conducting reviews, BT, and FT.
5	The interviewee is currently working in ST and CST. Before that the interviewee had various technical roles, including support and software design. Overall, she has over 10 years of experience in designing software.

Interview structure: The interview is structured in four

themes, namely: (1) interviewee knowledge and experience, (2) when and where to detect what type of defects, (3) how the tester conducts tests, and (4) strengths and liabilities of the current testing approach. The interview is semi-structured and consists of mainly open-ended questions, hence allowing to follow and discuss interesting issues, or change the order of questions. The interviews approximately took 90 minutes to complete. The contents of the interview were as follows:

- **Part I: Interviewee knowledge and experience:** This theme focuses on obtaining information about the testers knowledge and practical experience (professional background and education, explaining current role at the company, roles before current role).
- **Part II: Understanding test levels and their responsibilities:** This theme focuses on determining which types of defects are detected at which test level, using a defect classification scheme for telecommunication systems from Damm and Lundberg [9].
- **Part III: Understanding how testing is done and its related challenges:** This theme focuses on how testing is done and what challenges occurred, including characterization of test objects at different levels, input used to derive test cases and the quality of the input used (e.g. requirements), tool support, and general strengths and challenges observed.
- **Part IV: Closing:** The interviewees were asked whether they want to add something important that was not covered in the interview questions yet.

Before starting the actual interview, the purpose of the study and the reasons for the selection of the interviewee were provided. Furthermore, the interviewee was asked whether we were allowed to record the interview for transcription. The interviewee was also informed that all information collected was treated anonymously and will be aggregated with the information provided by the other interviewees.

Quantitative Data: With respect to defects we look at distribution of types of defects and defect criticality per test level, and fault slippage. The types of defects discovered at the test levels reveal which types of defects the test level is actually able to capture given the current test practices employed. Fault slippage measures indicate whether each testing phase is able to detect the defects it is supposed to detect (see [10]). The data is available through a company internal and proprietary defect reporting system. For this study we focus on recent defects (in the past 12 months) that reflect the current test strategy employed at the company.

3.4 Data Analysis

Interviews Audio recordings of all five interviews were transcribed. We used colour coding for initial data extraction from transcribed interviews where one unique colour was assigned to each one of the following:

- Challenge, problem, malpractice, limitation and missing information for testing
- Benefit or strength of current practices, tools or processes
- Current practice, way of doing work or tools used
- Improvement suggestions
- Definitions of terms, test levels and artefact descriptions

While colour coding, brief notes were made about the statements making use of the context of the statements and reducing misinterpretation later on. These colour coded statements and their brief descriptors were extracted and were put in a spread-sheet verbatim while maintaining traceability to the source. At this point these were assigned codes (according to Table 3) to capture their relation to the respective test level. A separate spread-sheet was created for each of the five items above. Next step was to aggregate the repeating statements which was done by repeating the following this process for all five sheets:
Step-1: For the first statement create and log a code.
Step-2: For each subsequent statement identify if a similar statement already exists. If it does log the statement with the same code, otherwise create a new code.
Step-3: Repeat Step-2 until the last statement has been catalogued.
A short description was given to each of the resulting clusters with same code. As traceability was ensured between the clusters, their summary/description, individual statements and audio recordings, second author was able to review the results of the process whether the statements were correctly clustered together. The disagreements were resolved by discussion.

Documentation was analysed through the same coding scheme as the interviews (as explained above).

3.5 Threats to Validity

While designing and conducting this case study various conscious decisions were taken to strengthen the validity of results. Using the checklist proposed by [22] we evaluated the case study protocol. This ensured that we had addressed all the critical requirements of case study design including aims of the study, defining the case, unit of analysis and data collection methods among others. Another researcher who has extensive knowledge and experience in case study research also reviewed the protocol. Furthermore, this detailed protocol was kept up-to-date, reflecting the actual course of the case study.

Construct validity: Both methodological (interviews and archival data analysis) and data source (practitioners and defect database) triangulation were used to strengthen the evidence generated in this case study. Using appropriate amount of raw data and through clear chain of evidence (maintaining traceability between the results, qualitative data and sources), this validity threat was minimized.

Internal validity: In this case study, the challenges were examined in the context of SoS, and the relation of SoS context to challenges was discussed. However, given the complexity of a real world organization and the fact that confounding factors are always a challenge when studying real world systems, the isolated effect of SoS characteristics can not be established, and requires further investigation.

External validity: There are too few empirical studies yet to make general claims, but the case in this study represents a typical complex SoS based software product development situation and is likely to apply to similar contexts w.r.t. system complexity, domain, etc.

Reliability: By involving more than one researcher, each actively engaged in the design, review and execution of the study, where case study protocol was the means of communication and documenting the agreement. Thus we believe that the resulting detailed protocol serves as a good means

to replicate this study. The detailed documented steps of data collection, processing and analysis also increase the reliability of this study.

4. RESULTS

The results present the current practices of testing SoS, the challenges and their relation to SoS characteristics, as well as SoS solution proposals.

4.1 SoS Testing Process and Practices (RQ1)

Test Process: We found that overall there are six Test levels at the case company. These were visible in both the test process documentation in the company and the interview results. Furthermore, the organization of testing teams and responsibilities is roughly organized around these levels as well. We found additional test levels in the defect database, however, based on the defect reports from the last two years these levels were not used at all in reporting. This hints redundancy of some of these levels. One explanation for no faults reported to "Regression test" is that regression testing is performed within other phases of testing and therefore is not considered a separate phase. Based on the congruence between the latest documents, interview results and defect reports we found six levels as shown in Table 3. These test levels are depicted in the overall development process in the Figure 2.

Table 3: Test levels in the case company

Code	Description of test levels
Rev	Reviews using static analysis and Visual "Diff" tools are performed by the development teams.
BT	Basic test is done in the Agile Team (AT) sprint by the development teams and uses implementation as the test reference. It includes unit testing with JUnit [1] and TestNG [4]. Input for tests are user stories, protocol specifications, component descriptions, dimensioning requirements, and traffic models.
FT	Function test focuses on testing isolated functions and is done in AT sprint by the development teams using Testing and Test Control Notation (TTCN) [3].
ST	System test tests the integration of the AT sprints (teams) in pre-defined cycles. Test cases created in FT and previous releases are used. No new test cases are written. The testing is done on a different branch along with regression testing using self-contained test cases (each test case sets up configuration, executes test scenario, and removes configuration once it is done). In addition, the company uses traffic models and simulates the system behaviour under load.
CST	Compound system test, tests the integration of systems into the overall system. Here, the emphasis is on making sure that the system functionality when integrated, still fulfils non-functional requirements (e.g. load), and is installable.
FOA	After system test is completed, the product is tested by installing it on a trial customer network. After this the product is launched to the general market.

Test Levels and their responsibilities: We used interviews to explore the practitioners' perception as to which defects are found at each of these test levels. The results are presented in Table 4. The first column depicts the fault classification used in the case company and the numbers in each row show how many of the five practitioners interviewed consider that this defect type is often found at this level. From the opinions expressed in the interviews most of the defects are found at ST. For certain defect types this is an expected

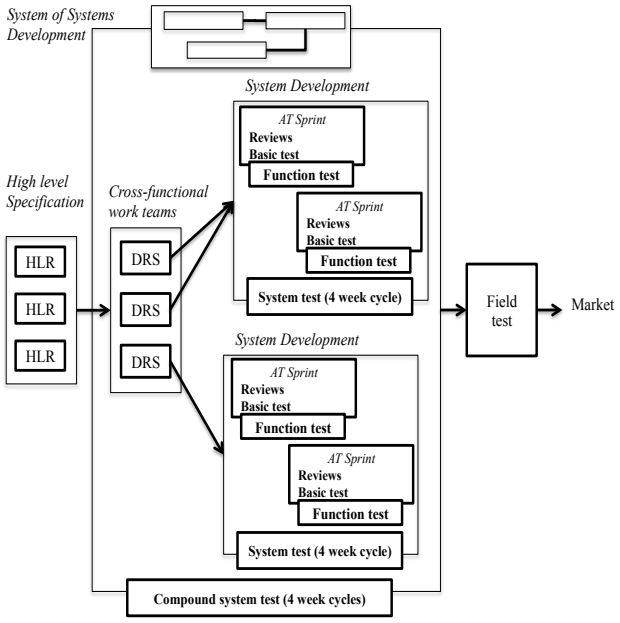

Figure 2: System of Systems development process with test levels at the case company

behaviour e.g. quality assessment at system level is not possible before this level. Perhaps that is why there is a consensus that Quality related issues (performance, robustness and concurrency) are found in ST. One practitioner said, *"the single biggest place where we find faults is in ST"*. Practitioners perceive that programming faults are discovered in basic testing which is an expected behaviour. Although reviews are recognised as a good practice and are used in the company but in practitioners' opinion it is not effective in discovering defects. Only one practitioner was satisfied with the effectiveness of current reviews.

Table 4: Fault Classes [9] and where Practitioners find them based on Interviews

	Rev	BT	FT	ST	CST	FOA
Internal Interface	2	2	2	2		
External Interface			2	2	1	
Human Interface			2	3	1	2
Coding Errors / Coverage	1	4	1	2		
Performance	1			5	3	
Robustness				5	2	
Redundancy				4	4	
Concurrency				5	3	
Configuration				3	2	2
Missing Functionality				2	1	1
Wrong Behaviour	1			1	1	1

Test Efficiency in the current situation: As a measure of test efficiency the company uses fault slippage, indicating whether a fault is found in a later test level than the one where it ought to be found. From Table 4 we observed that there is some fault slippage in the testing process. For example, consider the defects triggered from "internal interfaces" and "Coding errors" that should ideally be found in the early test levels like reviews or basic test. However, in practitioner's opinion these defects often slip through to later stages.

To assess the validity of these observations and look at the

effectiveness of testing quantitatively we used the FST data. We analysed the defect reports for the last calendar year since 01.01.2011- 01.02.2012 (see Table 5 for the results).

Table 5: Fault slip through data

'Should' to 'Did' detect	Rev	BT	FT	ST	CST	FOA	Total slippage
Rev	**30.76**	7.69	30.76	19.23	0.00	11.53	25.71
BT	*6.15*	**26.15**	32.30	18.46	4.61	12.30	62.85
FT		*7.14*	**50.00**	28.57	14.28		8.57
ST				**88.88**	11.11		1.42
CST					**100.00**		0.00
FOA							**0.00**

In Table 5 the values on the diagonal (in bold face) represent the percentage of defects that were found exactly where they are ought to be found. The percentage points in the last column "Total slippage" accounts for the percentage of slippage from each level compared to the total slippage from all test levels. In terms of overall slippage from one test level Basic test (with 62.85%) and Reviews (with 25.71%) have a lot of potential for improvement. From these results ST and CST are the top performers in terms of catching the right faults. As pointed out by one of the practitioners, *"The single biggest place where we find faults is in function test"* we can see that most of the slippage from Reviews and BT are caught in Function test (30.76 % and 32.30% respectively).

In Table 5 there are some unexpected values below the diagonal (in italics). There could be a number of explanations for it, the simplest one that it was a mistake in reporting the data or that although the test strategy states that the defect should have been found later but it was found in an earlier phase, thus requiring an update to the strategy or it was just a coincidence that this defect was found earlier and doesn't warrant a strategy update. However, we did not confirm or reject any of the explanations for this small percentage of defect reports.

We can see some congruence between results from Table 4 and Table 5. The majority of the defects are found in the later test levels especially in FT and ST. Top two levels with maximum slippage are BT and reviews respectively.

Table 6: Fault slip through to customer

'Should' to 'Did' detect	Percentage found by Customer
Rev	15.27
BT	63.88
FT	6.94
ST	11.11
CST	1.85
FOA	0.00
Customer	0.92

Looking at the problems found by the customer (as shown in Table 6) we can see that a high percentage of slip through is from Basic testing and Reviews. Having triangulated the practitioners' observations with quantitative data, we delved deeper to understand the practical challenges and their influence on these symptoms.

4.2 Challenges (RQ2)

From the interviews and defect databases we found three main undesired results with respect to outcome variables in testing, namely fault slippage, long turnaround time and maintenance of test suite. These outcomes are influenced by various issues and challenges, which were also identified from the interviews. In total we identified 30 different challenges that were mapped to three levels: 1) challenges that are not different in SoS context and other contexts, 2) challenges that are amplified in SoS context but that can also be found in other contexts, and 3) to new challenges specific only to SoS context (see Figures 3 , 4 and 5). We do not claim cause-effect relations, as those would have to be established in controlled environments. Hence, the relation between challenges and outcome variables should be seen as indicators in this exploratory study. The challenges in levels "2" and "3" as described above are also mapped to the characteristics of SOS (presented in Table 1) that aggravate or cause these challenges.

Figure 3, shows the challenges that influence the undesired result of fault slippage. In the related work (Section 2) it was highlighted that the independence of system development leads to challenges in collaboration. As there are many organizations integrating their interacting system and there are many levels of testing (see C1, C2, and C3 in Table 1), the challenge of unclear responsibilities of test levels is apparent. Collaboration challenges and independence (C1 and C2) also affect knowledge sharing, and hence lack of compliance to processes, lack of shared guidelines for testing at various levels and tools. It also leads to a lack of responsibility for test suits that are shared by different systems when they are integrated. SoS affects testability of requirements due to its complexity (C4), and that a requirement concerns multiple systems and in order to interpret them reasonably well a wide domain expertise across system boundaries is needed. There is also a lack of thorough analysis and selection of test cases, which is influenced by the SoS complexity (C4). Other influencing factors (e.g. early evaluation of quality attributes [14], time to market pressure and influence on test [6], difficulties in basic test, and lack of independent verification (people who code also write the test cases) [18] are rather generic challenges.

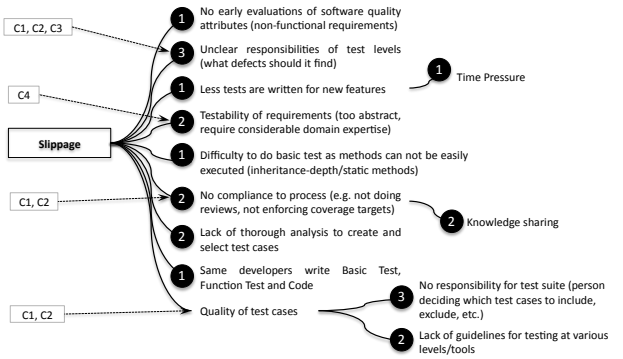

Figure 3: Challenges influencing fault slippage in the case company

In Figure 4 the challenges to the maintenance of regression suite are visualized. Because of the growing number, size and complexity of systems (C4) put together in the SoS (C3 and C5) it is imperative that the system test will get more and more test cases related to FT in regression suite. Furthermore, there is no formal responsibility for maintenance

of the test suite given characteristics C1 and C2, which also leads to no detection of redundant and obsolete test cases. The sheer amount of test cases added makes it a difficult problem to address. Any centralized solution is likely to be overwhelmed and with no shared standards (difficult to achieve in a SoS with managerial and operation independence) the current decentralized approach is not working, as the quality of test cases in terms of design, implementation and readability is a major issue for maintenance. Challenges such as test case readability (e.g. addressed to behaviour driven development [23]), company proprietary code due to lack of tool awareness, and lack of separation between test code and test data [12] are rather generic. However, non-compliance is aggravated in SoS due to difficulties in spreading news about good tools.

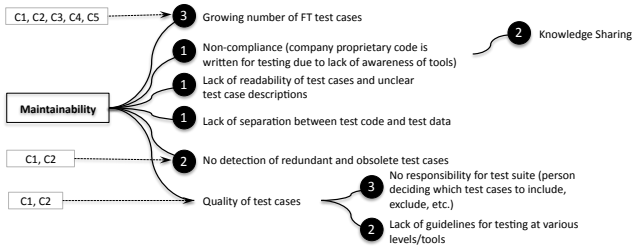

Figure 4: Challenges influencing maintenance of FT regression suite

The challenges contributing to a long turnaround time for regression test are shown in Figure 5. The number of growing number of FT test cases affects the maintainability of the test suite, and is significant due to the SoS context, as was discussed for Maintenance. The difficulty of doing basic testing leads to the misuse of the FT framework, which is amplified by the SoS complexity (C4) and lack of communication and interaction (C1 and C2) as the framework now includes many BT tests, that take more time to be executed due to the limitations of the FT framework. It should also be emphasized that many of these challenges influence multiple problems and have some interactions between them as well. The platforms inefficiency is one dimension of the problem because it is not easy to switch to a different framework because of the cost of migration of existing test cases and retraining the resources. Therefore the solutions to improve the turnaround time may take a multi pronged approach e.g. reducing the number of test cases to run by prioritization, selection or using the framework for its strengths. Another option is to think about smarter ways to design to specifically improve the turnaround time. Prioritization of test cases [25, 21] and TTCN as a test framework [26] have been addressed generically in research contributions.

4.3 Improvements (RQ3)

We identified 14 improvement actions in order to reduce/ mitigate the negative impact of the issues on fault slippage, maintainability of test cases, and turn-around time. The improvement suggestions have been organized to five groups based on which area they affect. The groups are developer quality assurance, function test, testing at all levels, requirements engineering and communication.

4.3.1 Developer quality assurance

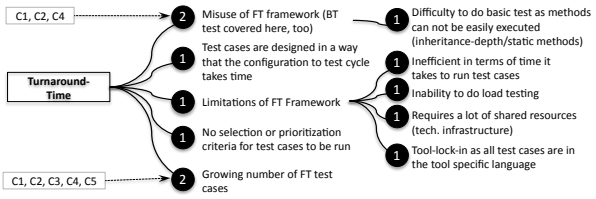

Figure 5: Challenges influencing the turnaround time of FT regression suite

Table 7: Improvement Ideas and Effected Testing Processes

Improvement ideas	Rev	BT	FT	ST	CST	FOA
Reviews	√					
Basic Testing		√				
Maintenance of FT reg suite			√			
Controlling size of FT reg suite			√			
Feedback time for FT reg suite			√			
Guidelines for FT tool usage			√			
Tech improvements to FT tool			√			
Improving test case quality		√	√	√	√	√
Definitions of test levels	√	√	√	√	√	√
Increase tool usage:	√	√	√	√	√	√
Early quality evaluations	√	√	√	√	√	√
Requirement testability			√	√	√	√
Feature status tracking	√	√	√	√	√	√
Improved interaction	√	√	√	√	√	√

The improvement suggestions in this group affect the work practices of individual developers and the results produced are not used outside of individual AT.

Reviews: Code reviews have helped in early detection of coding fault, however, there is a need for organization wide adoption of the practice. Reviews should be supplemented with tools that support visualization of code changes in the artefact under review. To utilize the human resources effectively, automated static code analysis should precede reviews. This will enable the reviewers to focus on the defects that the tools cannot find. These tools are generally easy to use and if configured properly can reduce the number of false positives. Thus, reviews can help avoid certain fault slippage and provide means for early evaluation.

Basic testing: Practitioners from development teams believe that increased basic testing has reduced the number of faults that slip through to the function test. They acknowledge that these improvements in basic test have been possible because of inclusive discussions with various development teams. This is a clear indication of why it is necessary to have stakeholder involvement and buy-in/consensus on decisions. Furthermore, some teams set and enforce coverage goals. They use code coverage tools, as these tools help to see the extent of testing. However, there is a need for wider adoption of these tools through out the company. Some practitioners also suggested that test-driven development and use of mock object techniques could offer avenues for improvement.

4.3.2 Function Test

The improvement suggestions in this group focus on the function test level.

Maintenance of FT regression test suite: There was a consensus on having persons with formal responsibility for maintenance of the test suite. This seems plausible, but

considering the size of the organization and amount of test cases it will require a lot of skills, effort and coordination between teams. However, the current practice of having a distributed mechanism has not really worked either. Perhaps in the SoS context the only viable solution is that of shared responsibility with some central governance. Thus, having multiple levels of control on what is contributed to the regression suite. For example, first the individual teams assess the quality of FT and suggests an inclusion to the suite, and then the tests are only included if the person(s) with formal responsibility approves them. Furthermore, improving the readability of test cases and enforcing design principles when developing test code will reduce the maintenance cost as well.

Controlling the size of FT regression suite: By improving the effectiveness of reviews and BT (see suggestions in Section 4.3.1) the unnecessary load on FT will be reduced. Similarly having responsible person(s) for test suite maintenance will give an opportunity to have a filter on new test cases being added. Similarly such person(s) may also review existing test cases and remove redundant or obsolete test cases from the suite, thus reducing the number of test cases.

Reducing the feedback time for FT regression suite: Practitioners believe that having prioritization or selection criteria will bring significant improvement to feedback time for the regression suite. So rather than running all the test cases all the time, two suggestions for prioritization criteria were: to run the most relevant tests first or to run the tests that fail frequently before others in the suite. Another step in the right direction is that the test cases have been tagged at a high-level based on the protocols and features they test. This is useful to select relevant test cases and to identify which parts of the system are being neglected in testing and which other parts have a high overlap in test cases. Removal of redundant and obsolete test cases and using the framework for FT (instead of BT) will also alleviate the problems aggravated by having a large regression suite in ST where all the FTs are added, as well as the problems related to an inefficient testing framework.

Guidelines for FT tool usage: We found that TTCN, i.e. the tool used in FT, is sometimes misused for unit testing which is not the strength of the framework. In large organizations dealing with SoS there is a Lock-in with respect to tools and it is very costly to switch to a different tool, especially when it is as embedded as TTCN. Therefore, the practitioners suggested to use TTCN for FT only as it has following strengths:

- Ability to test multiple protocols: is a major strength of TTCN over other testing frameworks.
- Automation helps save time: with use of TTCN for testing the time taken for test has reduced considerably. As one practitioner said, *"For us, for instance if we want to test a Product Customization, with say 30 test cases. Just executing them may be takes about 15 minutes. But doing it manually will take may be 3-4 hours"*.
- Useful for protocol verification: the ability to manipulate at bit level gives better control of protocol messages and parameter verification. This makes it very useful for external interface testing.

Thus, by taking steps in Section 4.3.2 to manage the suite size we can use the resource intensive TTCN for the testing that cannot be tested without it.

Technical improvements to FT tool: Many of the maintenance problems of the FT suite stemmed from the technical shortcomings of the TTCN tool used in the company, thus various improvement ideas for it were suggested in the interviews. Practitioners had complaints about the inability of TTCN to support load and GUI testing. There is also a need to improve the TTCN editor to identify multiple test headers in the same file and improve the readability of test cases. It should be easy to write TTCN test cases without having to repeat a lot of things, and to write configurations in the test cases so that verification is facilitated. One suggestion was to develop wrappers or high-level language constructs for setting up complex configurations, which will reduce the requirements on knowledge about subsystems and intricate details of how the protocols work. This will not only make writing test cases easier, which means even under time constraints it is still possible to develop test cases without the threat of mistakes in configuration.

4.3.3 Testing in all levels

The improvement suggestions in this section affect testing at all levels of testing from basic testing (BT) to field tests (FOA).

Improving the test-case quality: Having templates and guidelines for writing test cases particular to various test levels and tools used will help improve the quality of test-cases. This will enforce a consistent structure in test-cases and help teams following best practices to enabling readability, good design and high coverage. Furthermore, improving the quality of test cases will significantly improve the confidence in the testing suite's ability to find faults. The current mechanism of writing configurations at the bit level results in test cases that are error-prone. In the complex SoS context it is almost impossible to have such detailed knowledge about all the systems. Thus, having a wrapper (which also addresses the technical issues of the FT tool) to provide abstraction for the low level configurations will facilitate writing test cases and reduce the necessity of in-depth domain knowledge about various systems in SoS. Having test-case templates, documented best practices, and high-level constructs to specify test cases will also alleviate the effects of employee turnover as the knowledge will be embedded in the artefacts. Furthermore, reducing fault slippage by writing tests in pairs and/or review test cases by another team member will reduce chances of misinterpretation of requirements.

Definitions and responsibilities of test levels: Practitioners identified the need for better organization wide definitions of test levels with clear strategies of what should be verified at each test level. They also think that a better analysis of what we want to verify in each test level can reduce the overlap in testing. It is important to make a trade-off between early detection and cost of set-up. One may identify certain faults in basic test but the cost of setting up the environment to test may be too high. It is an encouraging sign that the testers understand this e.g. while commenting on the role of basic testing one practitioner said, *"We are aware that we can't test everything and may be we shouldn't in basic test"*.

Increase tool usage: Practitioners were aware of many improvements that could be gained by wider usage of test

tools. For example, generating tests instead of manual coding, while talking about the benefits of a tool used to create trees and simulation test cases the practitioner said, *"then you can run a lot of logic in the tree without writing in TTCN"*. Similarly, use Mock tools as suggested in Section 4.3.1 instead of statically programming the interfaces. Also, it was suggested to use tools to generate test data that helps in e.g. boundary value analysis, thus maintaining a differentiation between test code and data.

One practitioner highlighted the importance of tools that can provide a management view by summarizing and presenting data to support decision-making. This aligns well with the need of technical overview in a SoS. SONAR [2] is one such tool that can help providing:

- a management view of measurements and status.
- feedback on test cases, where there are less test cases or too many.
- and highlight, which test cases are not designed properly.

Early quality evaluations: The practitioners expressed the need to run non-functional tests sooner than currently possible. One solution is to look for performance issues that can be identified earlier without running the non-functional tests. This could be done by reviews or static code analysis to e.g. design and complexity of algorithms, memory management that is likely to result in performance degradation. Yet another complementing solution is to have early architectural evaluations for quality requirements e.g. by using prototyping.

4.3.4 Requirements engineering and communication

The large size of the SoS context can easily lead to communication gaps. Furthermore, requirements form the bases against which tests are developed. Thus, it is natural that practitioners suggested how improvements in requirements engineering and communication would help the case company.

Improving the testability of requirements: In a typical case in SoS context, requirements are written to describe a system service that involves multiple systems. It therefore helps when the feature lead (person owning a feature) does a pre-analysis of requirements to understand which systems will be affected and helps testing teams to anticipate workload. Similarly, having concrete user-stories that describe one standard scenario and not the whole service is useful for testing. It should describe a standard scenario with internetworking description (the interactions with other systems) without delving into everything that can go wrong. Another suggestion is that the requirement owner should write function test on at least external interfaces so that the requirement will be documented in a testable manner.

Feature status tracking: There should be a mechanism in the development process to document and communicate the decisions regarding consciously delayed features as it otherwise will create false positives in testing. Currently, the test team creates a defect report for missing functionality and it goes through an expensive process to handle them. Furthermore, there is no distinction between unintentionally and intentionally missing features.

Improved interaction between teams and cross-functional teams: System management is in between the customer and design/test teams and they have a good understanding of the customer needs. So they can be helpful to ask for missing information or explanation to avoid misinterpretation.

Similarly, a new initiative to involve ST teams early on in the process was welcomed by the practitioners. Now the testing team has a better understanding of features and can estimate the expected workload better. This early involvement has led to improved planning and reduced the time it takes the test team to understand the deliverables. If we have a cross-functional team then there will be opportunities to have early deliveries to test and have feedback sooner about problems and avoid expensive opinion/question defect reports.

5. CONCLUSIONS AND DISCUSSION

We made four contributions in this paper. First, we presented a study that focuses on testing challenges in SoS context. We mapped the challenges we found to three categories, 1) challenges that are not different in SoS context and other contexts, 2) challenges that are amplified in the SoS context but that can also be found in other contexts, and 3) to new challenges specific only to SoS context. To our knowledge, only Columbi et al. [7] has addressed this topic previously. In comparison to their SoS challenges we can say that their challenges (2) and (4) in Section 2 are not true in our case as the case company has tests to evaluate overall SoS capabilities and overarching test scenarios in compound system tests and field tests (see Figure 2). Their challenge (1) is partially true in our case as the current organizational structure does not support testing in SoS context, for example the lack of regression test suite maintenance can be traced back to the organization of our case company. Finally, their challenge (3) indicating that SoS systems have a high number of requirements leading to a high number of tests is completely true in our case. Reflecting their solutions proposals with the light of our case we can state that their proposal (2) using risk assessment to prioritize testing would be useful in our case as well. Their solution proposal (3) suggest focusing tests on interfaces might be applicable in our case but we think that other test focusing criteria should be used as well. Their solution proposal (1) "build little, test little" has already been applied and even exceeded in our case as the testing is following agile software development pace and testing with high coverage is performed continuously on different levels (see Figure 2). Their final proposal (4) to provide designated environments and people for integration does not seem that useful in our case as testing integrations did not seem to be a big problem and on the other hand designated people craved more support to regressions test maintenance rather than integration test.

Second, we found a contradiction between the software process improvement focus areas stemming from the fault slippage measure and interviews with practitioners. The fault slippage of internal processes and from customers clearly pointed to improving the areas of review and basic test that resulted in the highest fault slippage numbers (see Tables 5 and 6). However, the challenges and improvement ideas from practitioners pointed mostly to improving the functional tests and the system test suite that had poor maintainability and turnaround time, but high fault detection capability (see Figures 4 and 5 , and Table 7). On the other hand the reasons for problems in FT were caused at least partly by the fact that people did not do basic testing, but implemented the same test as functional tests because BT

offered a poor technical support for testing. In the end, we cannot be certain if the company is better off improving functional or basic tests. We believe this can be generalized to the questions whether companies are better off in improving the practices that are already strong, but still have good improvement potential, or the practices that are weak.

Third, we found an interesting circular relationship between the maintenance and turnaround time of functional test. It is difficult to improve turnaround time if the test code has low maintainability. However, the low maintainability also increases the turnaround time when developers add new test cases rather than modify the existing test cases to complement for new features. We believe that circular relationship can be attributed to the SoS context and the managerial independence that each development team has.

Fourth, we found that test case maintainability and maintenance are a big problem in SoS context. Future work should see how could the techniques and practices for regular software maintenance help test code maintenance in SoS context.

6. ACKNOWLEDGMENTS

This work has been supported by ELLIIT, the Strategic Area for ICT research, funded by the Swedish Government.

7. REFERENCES

[1] JUnit: A programmer-oriented testing framework for Java. `http://www.junit.org/`. [Acc. Mar. 2012].

[2] Sonar - an open platform to manage code quality. `www.sonarsource.org`. [Accessed Mar. 10, 2012].

[3] Testing and Test Control Notation Version 3 (TTCN-3). `http://www.ttcn-3.org/`. [Acc. Mar. 2012].

[4] TestNG - a testing framework. `http://testng.org/doc/index.html`. [Acc. Mar. 2012].

[5] M. C. B. Alves, D. Drusinsky, J. B. Michael, and M. T. Shing. Formal validation and verification of space flight software using statechart-assertions and runtime execution monitoring. In *Proceedings of the 6th International Conference on System of Systems Engineering*, pages 155–160. IEEE, 2011.

[6] J. Christie. The seductive and Dangerous V-model. *Testing Experience*, pages 73–77, 2008.

[7] J. Colombi, B. C. Cohee, and C. W. Turner. Interoperability test and evaluation: A system of systems field study. *The Journal of Defense Software Engineering*, 21(11):10–14, 2008.

[8] J. Dahmann and K. Baldwin. Understanding the current state of us defense systems of systems and the implications for systems engineering. In *Proceedings of the 2nd Annual IEEE Systems Conference*, pages 1–7. IEEE, 2008.

[9] L. O. Damm and L. Lundberg. Identification of test process improvements by combining fault trigger classification and faults-slip-through measurement. In *2005 International Symposium on Empirical Software Engineering, 2005*. IEEE, Nov. 2005.

[10] L.-O. Damm, L. Lundberg, and C. Wohlin. Faults-slip-through - a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006.

[11] DoD. Systems and software engineering. systems engineering guide for systems of systems, version 1.0. Technical Report ODUSD(A&T)SSE, Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Washington, DC, USA, 2008.

[12] M. Fewster and D. Graham. *Software Test Automation*. Addison-Wesley Professional, Sept. 1999.

[13] R. A. Gougal and A. Monti. The virtual test bed as a tool for rapid system engineering. In *Proceedings of the 1st Annual IEEE Systems Conference*, pages 1–6. IEEE, 2007.

[14] J. E. Hannay and H. C. Benestad. Perceived productivity threats in large agile development projects. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2010)*, 2010.

[15] J. A. Lane and R. Valerdi. Synthesizing sos concepts for use in cost estimation. *Systems Engineering*, 10(4):297–307, 2007.

[16] G. Lewis, E. Morris, P. Place, S. Simanta, D. Smith, and L. Wrage. Engineering systems of systems. In *Proceedings of the IEEE International Systems Conference (SysCon 2008)*, 2008.

[17] G. A. Lewis, E. J. Morris, S. Simanta, and D. B. Smith. Service orientation and systems of systems. *IEEE Software*, 28(1):58–63, 2011.

[18] R. O. Lewis. *Independent verification and validation [Elektronisk resurs] : a life cycle engineering process for quality software*. Wiley, New York, 1992.

[19] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[20] K. Petersen and C. Wohlin. Context in industrial software engineering research. In *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, pages 401–404, 2009.

[21] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: An empirical study. In *Proceedings of the International Conference on Software Maintenance (ICSM 99)*, pages 179–188, 1999.

[22] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[23] C. Solís and X. Wang. A study of the characteristics of behaviour driven development. In *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2011)*, pages 383–387, 2011.

[24] R. K. Yin. *Case study research: design and methods*. Sage Publications, Thousand Oaks, 3 ed. edition, 2003.

[25] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 2010.

[26] B. Zeiss, H. Neukirchen, J. Grabowski, D. Evans, and P. Baker. Refactoring and metrics for ttcn-3 test suites. In *Proceedings of the 5th International Workshop on System Analysis and Modeling: Language Profiles (SAM 2006)*, pages 148–165, 2006.