31st Annual INCOSE international symposium
virtual event
July 17 - 22, 2021

Challenges in Detecting Emergent Behavior in System Testing

Kent Aleksander Kjeldaas
Kongsberg Defence and Aerospace AS
Kirkegårdsveien 45, P.O. Box 1003 3601
Kongsberg, Norway
+47 41 22 98 33
kent.aleksander.kjeldaas@kongsberg.com

Rune André Haugen
Kongsberg Defence and Aerospace AS
Kirkegårdsveien 45, P.O. Box 1003 3601
Kongsberg, Norway
+47 93 43 14 53
rune.andre.haugen@kongsberg.com

Elisabet Syverud
University of South-Eastern Norway
Hasbergsvei 36, 3616 Kongsberg, Norway
+47 41 36 04 88
elisabet.syverud@usn.no

Abstract. System integration testing in the defense and aerospace industry is becoming increasingly complex. The long lifetime of the system drives the need for sub-system modifications throughout the system life cycle. The manufacturer must verify that these modifications do not negatively affect the system's behavior. Hence, an extensive test regime is required to ensure reliability and robustness of the system. System behaviors that emerge from the interaction of sub-systems can be difficult to pre-define and capture in a test setup using acceptance criteria. Typical challenges with current test practice include late detection of unwanted system behavior, high cost of repetitive manual processes, and risk of release delays because of late error detection.

This paper reviews the state of practice at a case company in the defense and aerospace industry. We use an industry-as-laboratory approach to explore the situation in the company. The research identifies the challenges and attempts to quantify the potential gain from improving the current practice. We find that the current dependency on manual analysis generates resources -and scheduling constraints and communication issues that hinder efficient detection of system emergent behavior. We explore two approaches to automate anomaly detection of system behavior from test data. The first approach looks at anomaly detection in a top-down approach to give an indication of the system integrity. The second approach uses anomaly detection on system parts, resulting in the ability to localize the root causes. The work lays the foundation for further research of automated anomaly detection in system testing.

Introduction

Defense. Defense projects are notorious for cost and schedule overruns (Lineberger & Hussain 2016, Hofbauer 2011). Both high technical complexity and the supply chains issues contribute to the problem. (Lineberger & Hussain 2016).

Global supply chains and government negotiations have increased the complexity of defense system architectures. Defence systems uses commercial off-the-shelf components that companies around the world produce. This makes it important to specify correct interfaces and ensure commonality of the sourcing approach (Sols 2014). In addition, the supply chain negotiations of military developed products sometimes take place on a government-to-government level. This can cause the defense industry having limited control over the information and acquisition of several of its system parts. Projects will have to deal with uncertainty in the interfaces, functionality, and performance of the system parts while developing their system.

Company. A case study within a major defense and aerospace contractor, henceforth called the company, is the base for this research. The company has seen an increase in demand for missile systems. Customers demand projects with a shorter development time. The company desire methods that shorten the development process and reduce the potential for cost and schedule overruns.

Development projects in the missile industry typically range from 10 to 15 years. The life cycle for such systems typically spans decades. The long development time and lifetime requires foresight and long-term planning to keep up with technological advancements. The long development time requires an extensive testing regime to verify the system behavior throughout the lifetime.

Missile systems have complex system behavior and are "one-shot" systems that potentially have been stored for several years before use. This demands a high level of reliability and robustness in the test system. Systems Engineering practices help us understand the effects of complexity in our system.

Problem Statement. The company is experiencing a problem with undetected unwanted behavior during system integration tests. The current test procedures do not easily detect these types of system anomalies. The company typically discovers errors late in the development cycle or potentially not at all. Late detection gives little time to correct errors prior to product release. The errors vary from integration errors to unwanted complex emergent behavior. In this paper, we study the analysis of system test data to detect unwanted system behaviors that emerge from system modification.

Figure 1 shows the test workflow. The test process is semi-automated. The project team responsible for the system modification plans and assigns the tests. The test team schedule and execute the test, produce the data, and convert the test data to logs. The test team saves the logs for future reference. The test engineers run the test according to protocol, and the system meets the specified acceptance criteria. Analysts find and report critical errors to the development team if the test fails.

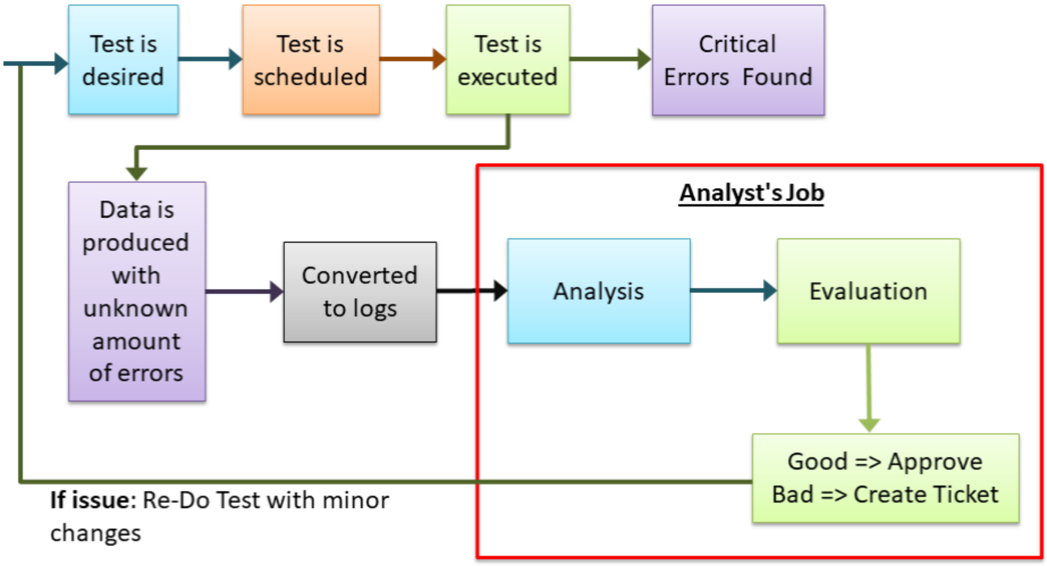

Figure 1. Test Workflow

By analyzing the data captured during scenario tests at system level, the company can detect unwanted systems behavior not captured by the test procedure. Therefore, the project development teams typically request analysis and evaluation of the approved test cases. Hence, the test engineers send the logs to an analyst for review and detection of potential anomalies. The analysts are experts in their respective sub-system parts, and they manually study and evaluate the test data captured during tests. The analysts use their tacit experience and system knowledge to detect emergent behavior by looking for abnormalities in the data.

Seen from the system-level perspective, each additional system part increases the chance of an error and anomaly checks are important to ensure reliability and robustness of the overall system. From the analyst viewpoint, the chance of finding an error in the data is low. If the analysts find an error, the system developers are not happy about the additional work needed to fix the error, nor the time delay it causes. This may lead analysts to prioritize the analysis work lower than other pressing tasks.

System tests sometimes detect unwanted system behavior at the overall system level. The system parts can embed the root causes of such system behavior, making it difficult to trace. Successful detection of the root cause may involve multiple experts analyzing lots of information to locate the sub-system causing the error.

It is important to find a proper balance when capturing test data for manual analysis. We want to minimize the overall data required to analyze and not to overload the analysts with unnecessary data. However, it is important to capture the right data at the right frequency to get sufficient information to understand the errors. It should be an iterative process to adjust the log data to fit the analysis need at different levels (system, sub-system, and component).

The company seeks to automate the manual process to limit the dependency on system experts and remove the bottleneck with manual analysis. The company expects that increasing the amount analyzed will increase the probability of early detection of unwanted emergent behavior. By automating the analysis, the company seeks to use the full potential of the data.

Research Questions. In this paper, we study the detection of emergent behavior during system integration testing. We focus on the practical challenges of detecting emergent behavior in complex systems. We differentiate between the detection of emergent behavior that we can do prior to the system integration phase and the detection of behavior that we can only do in hindsight. We attempt to identify the existing challenges, potential gains, and process improvements that help the detection of emergent behavior in complex systems.

Specifically, this research focuses on the following research questions:

- What are the current challenges in detecting emergent system behavior?
- How are cost and resource constraints affecting the detection of emergent system behavior?
- How can the company improve detection of emergent system behavior during the system integration phase?

This paper begins with a brief background of relevant topics. Next comes a presentation of the research method. Then, the paper explains the state of practice as found in the literature, the company's approach, and the challenges discovered. The next section explores approaches to the challenges. Finally, the paper discusses the suggested approaches before presenting the conclusion.

Literature Review

Systems Integration. System Engineering uses partitioning to reduce the complexity in systems. It divides the problem into several sub-problems that are easier to deal with. This process repeats until the problems are manageable. The System Integration (SI) process assembles all the parts to deliver the intended behavior and functionality (Sols 2014). System Integration is a front-loaded process that comprises creating acceptance criteria from the requirements of the system and criteria for the specific design. System integration tests use acceptance criteria as a measure to verify the system. Developers plan acceptance criteria into test cases. The test environment executes and verifies the test cases. An integration plan manages the assembly and test process. The company conducts top-level system testing after assembling all the system parts. While integration happens bottom-up, the integration strategy does not have to follow the same pattern. Typical strategies are bottom-up, top-down, and big bang integration. (Sols 2014).

Automated Testing. System Engineering uses test cases to qualify the system, and automated testing is automating these. (Engel 2010). Test automation is common for software components (Kolawa & Huizinga 2007), but we can also find it for other interdisciplinary components and subsystems. The goal is to save time by avoiding repetitive and manual check processes (IEEE Automated Testing 2000).

Anomaly Detection. Anomaly Detection aims to identify outliers in the system-under- observation. These outliers are anything that could be abnormal. Developers look for abnormalities that can give a sign if something is wrong with the behavior of the system. All anomaly detections require some sort of data as input. Developers can feed input in advance or the algorithms can learn-over-time. They can then apply statistical approaches to the data, enabling the algorithm to “learn” based on the input given. There has been a broad range of methods and practices for anomaly detection, capable of both detecting behavioral changes on the component level and the system level. Many of these methods utilize machine learning to detect anomalies (Thyago et al. 2019). Some applications of anomaly detection are detection of bank frauds, health monitoring, and predictive maintenance.

Complex Systems. Wolf & Holovet (2004) identified four categories for systems. These four categories are known, knowable, complex, and chaotic. Known systems have cause-and-effect relations that are straightforward and predictable. Through partitioning and detailed analysis, we can understand that knowable systems have cause-and-effect chains. While known and knowable systems have emergence that can be understood in advance, complex systems and chaotic systems do not have this perceivability and predictability even with methods such as reductionism and detailed analysis. This unpredictability results in the system exhibiting emergence.

Emergent Behavior. Emergence is a heavily contested term, used by several domains (Mittal et al. 2018). Wertheimer, Koffka, and Kohler explained emergence as `the whole being different from the sum of its parts`. While they stated this within the field of psychology, it holds true for system development as well. Complex systems often exhibit behavior that we cannot explain by only examining and understanding the behavior of the system parts alone. (Checkland 1999, Johnson 2006, Mogul 2006). The change in the behavior on the system-level compared to the system-parts explains why it is not enough to test the system parts individually but also test the system. A test regime that focuses on detecting emergence increases in relevance as systems become more complex, because of how emergence increases when systems grow in complexity in terms of interactions and potential states. (Bedau 1997, Holland 1999, Johnson 2006, Mogul 2006).

Emergence has been in the focus of researchers for several decades (Bedau 1997, Holland 1999), but few methods exist for the identification and detection (Seth 2008, Kubik 2003, Szabo & Teo 2012, Brown & Goodrich 2014, Mittal et al. 2018). With system complexity on the rise, alternative methods to detect, understand, and manage the increasing occurrence of emergence in systems are required (Mittal et al. 2018).

Fromm (2005) and Holland (2007) have identified several categories of emergence focusing on the feedback types and relationships between weak and strong emergence. Their work has since been supplemented by Rainey and Tolk (2015) and Mittal et al. (2018), ending up with simple emergence, weak emergence, strong emergence and spooky emergence. In short, simple and weak emergence is detectable and understandable in advance of testing, while strong and spooky emergence is not.

Complex systems can be understood only in retrospect and do not usually repeat, while complicated systems can be understood by reductionism and detailed analysis. Strong emergence is unpredictable and inconsistent in simulation, while weak emergence is predictable and consistently reproducible in simulation. The question whether simulation systems can generally reproduce strong emergence is not yet answered (Mittal et al. 2018).

Detecting Emergent Behavior. Detection of emergent behavior can happen at the top-down (macro-level) or the bottom-up (micro-level). If analysts detect emergent behavior on a macro-level, they must find a connection to the micro-level to identify the root causes. We can classify approaches in system testing in two orthogonal perspectives, detection during test execution and post-test detection (Kubik 2003). Both approaches require knowledge about the system to capture the anomalies, but live testing focuses more on “what” they achieve rather than “how” and hence requires less prior knowledge (Szabo & Teo 2012). Current methods focus on detecting the anomaly and leaving the evaluation up to the analysts. Metrics such as interaction (Chan 2011), statistical complexity (Shalizi 2006) and Hausdorff distance (Huttenlocher et al. 1993) have showed being able to capture emergent behavior.

Research Methodology

The research uses mixed methods, containing both quantitative and qualitative research methods to achieve this. The research takes place in an industry-as-laboratory environment.

Industry-as-Laboratory. This research uses the approach as proposed by Potts (1993), where an actual industrial setting is a test environment. The advantage with such an approach is the realism introduced into the research. Practical challenges in the theoretical frameworks can be hard to detect, unless one observes how it plays out in reality. One challenge emerging from such an approach is the potential of noise from company-specific problems that we cannot directly link to the research conducted.

Research Approach. This research starts by taking an inductive approach, by exploring the problem, gaining an understanding of the situation, and eventually coming up with a hypothesis. We structure the research into two parts, the first part comprising understanding the problem domain, and the second part consisting of exploring solutions. Both the understanding and the exploration parts each have their own evaluation, to verify the understanding and the explored solution.

At the beginning of the research, the researchers explored the current process in the company and looked at the strengths with the current process and the potential for automation. With the learning outcome from this process, the researchers continued with identifying the stakeholders. The stakeholders identified were the following: system engineers responsible for their own sub-systems, senior engineers responsible for a component, test managers and testers. The research conducted informal unstructured interviews with the identified stakeholders. The interviewer chose open-ended questions due to its exploratory nature. The researchers chose an iterative approach by first listening to the stakeholders before digging deeper into challenges that seemed to have a high impact. The researchers continued by studying how these challenges corresponded to the current process, and how the current process is structured compared to theoretical approaches for system integration.

The authors then created interviews to quantify and confirm challenges about the detection of emergence. The researchers divided the interviews into a quantitative and a qualitative part. The quantitative part aims to quantify the challenges previously discovered. The quantitative part focused on detecting new challenges not previously discovered. The authors conducted the interviews in the company. The researchers interviewed 15 stakeholders from different interdisciplinary teams face-to-face. The distribution of the domains interviewed includes software (3), GNC (3), system engineers (top-level (1) and sub-system-level (2)), hardware (3), testers (2), FPGA (1). The interviews employ a two-part interview format where the first part is highly structured, and the second part is semi-structured. The first part consists of background information, workload, and process specific questions while the second part consists of challenges and improvement areas with the current process and frequency and type of errors discovered during analysis. The goal of the interviews was to understand the current situation, specific challenges with the current process, and categorizing what data is relevant for detecting system-level errors. The highly structured part captures the workload and potential missing links in the process. The semi-structured part focused on capturing both specific challenges with the process and data that relates to the detection of system-level errors.

After the interview, the research split into two new directions. The first direction consists of creating a new framework attempting to solve the issues discovered. The second direction focuses on optimizing and improving the existing processes. The researchers created a proof-of-concept and fed the lessons learned back to the suggested frameworks for detecting emergent behavior.

Finally, the researchers conducted a new interview to evaluate the framework and the assumptions made. The researchers conducted structured interviews of eight analysts from various groups. The focus of the interviews was to discover the potential gain with a new framework, looking at the current cost of manual analysis, in terms of hours and lead-time. The researchers supplemented the interview findings with data gathered from company databases. The data explored consist of test information from the last 6 months and company registered issues going back 5 years. Last, we analyzed the feedback and introduced changes to the framework.

State of Practice

This research started by focusing on understanding the problem and connecting it to the current process and the current body of knowledge.

State of practice. Figure 2 shows a high-level view of the product development phases and the related integration activities. The state of practice consists of creating well-defined, quantifiable acceptance criteria that can verify the requirements and specific design choices. The company plans the acceptance criteria into test cases before execution and verification in the test environment. Manual analysis is a method used to supplement the test cases method. This typically consists of producing readable graphs and different views out of the test data, and has analysts look at the data for abnormalities. The project phases are iterative, and the project will go through integration at several points in the systems development. The integrations can be additions of new functionality, new capabilities, and new hardware. Each new integration will need verification with a mix of test cases and manual analysis. This repetition for each integration has led to automation of many test cases.

Company process. Figure 3 shows the platform development. The first goal is to get a system capable of executing high-level scenarios. The aim is to test on the system-level as soon as possible and ease the integration of new components and functions towards the system. During system-level tests, information about the interfaces and interactions are learned. Developers can use this information to improve the system in upcoming integrations.

The scenario dictates the stage and objective of the system. System engineers create scenarios with the goal of connecting as many test cases as possible. The execution of scenarios gives feedback to the testers in real time on how the system executes the scenario. An example of information provided to testers is the route taken versus the route planned. An extensive amount of information is stored for post-processing. The testers share the test results and the data when the test has finished, giving analysts a chance to analyze the data and provide feedback to developers.

In the company, the closer one is to the domain level, the more automation exists. Interdisciplinary systems seem to lack an automated continuous testing-regime for higher-level system parts. The software domain and software heavy systems deal with this better. The consequence of having manual testing at higher levels of the system is a resource intensive analysis process that also yields slow results. Slow results make long development-cycles that make it hard to re-engineer system parts and functionality while they are fresh in mind of the developers.

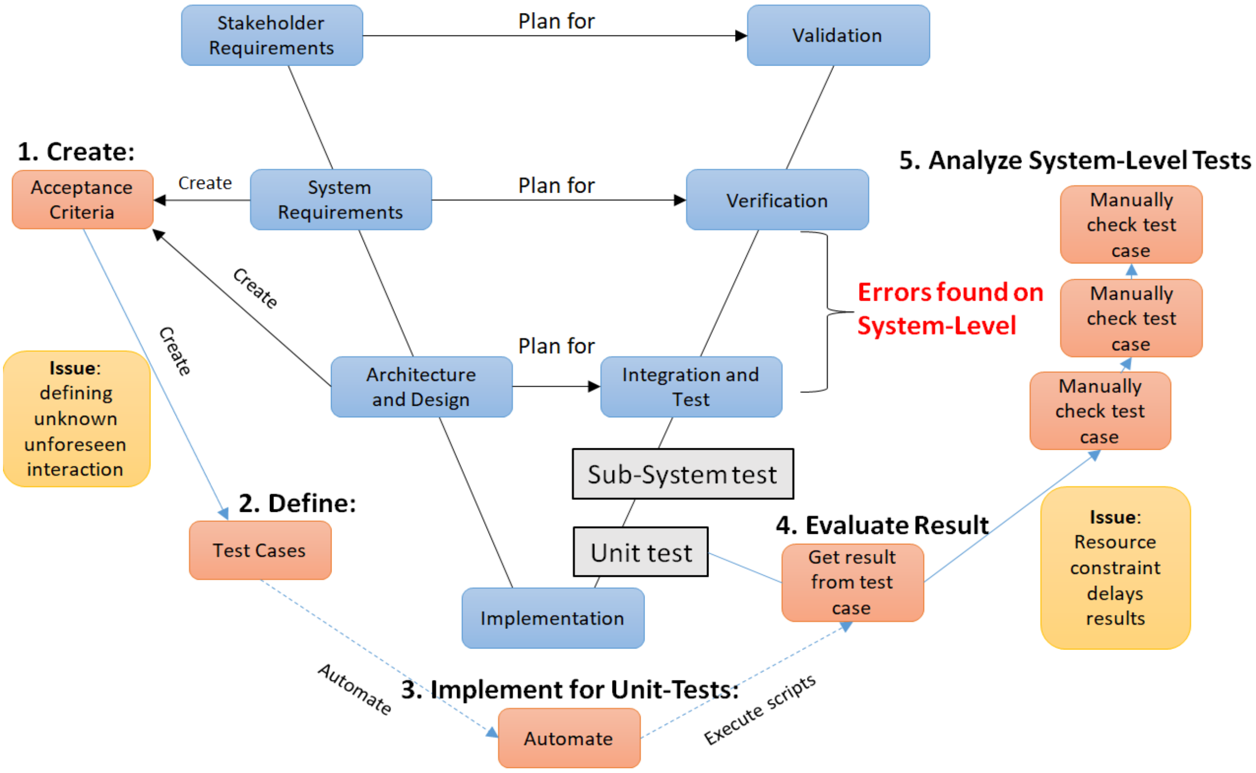

Figure 2. Product Development Phases and Integration Activities

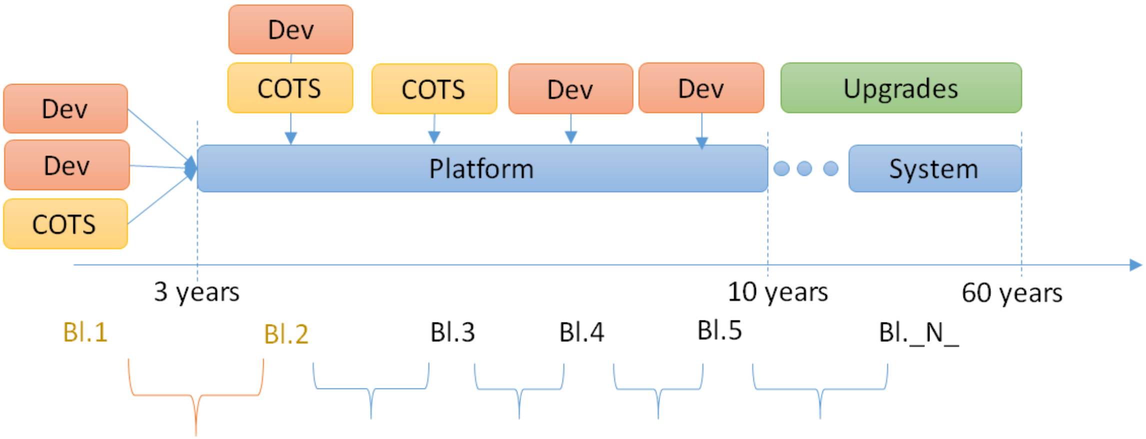

Figure 3. Platform development

Challenges in Detecting Emergent Behavior

Investigation into the problem domain. We base our findings on interviews with stakeholders, observations, and research into company data. The researchers conducted several informal interviews to discover the stakeholder's root causes and challenges with the current process. The first interviews aimed to explore the problem domain and the challenges experienced with detecting emergent behavior. The interviews uncovered two main root causes, one process specific and the second one being scheduling and communication specific. The researchers conducted two additional interviews. The first aimed to verify the assumptions made in the previous interviews and understand the importance of the different challenges discovered. The last interview focused on quantifying the potential gain.

Bottleneck. During this research, it was uncovered that the company only analyzes data under the following events:

- Addition of new functionality to the system (Integration Phase)
- Test shooting (Verification Phase)
- Handover to customer (Validation Phase)

Although analysts analyze tests when developers have added new functionality to the system, analysts only verify the new functionality itself and not all other potentially affected parts. The events above relate either to verify new functions added or to the entire system. The focus is not to detect errors but to verify the integrity. The rationale for only checking the data for verification is because the process is deemed too resource intensive. Table 1 shows the challenges discovered because of the resource constraint.

Table 1: Resource Constraint Challenges Identified

Challenges	**Description**	**Findings**	**Consequence**
Non-Continuous Testing	The System Testing workload is high in hectic-periods and almost non-existing in non-hectic-periods.	An average of 39% of the month goes to analysis work in hectic months, while only an average of 4% of their time in non-hectic months.	Late detection
Lack of Time	Work hours required to conduct manual analysis with current test-level	With an average of 50 tests executed per month: 1575 hours per month In high-intensity months, one can expect an increase due to additional tests being conducted (this should be within 25% of the time proposed)	Uncertainty and risk
	The analysts feel they do not have enough time for analysis	80% of analysts want to do more than what they feel they have time for (12/15) 60% of the analysts also desire to use more time than what they are currently using (9/15) 47% of the analysts use more time than what they feel they have available (7/15)	
	The analysis is important, and prioritized	Importance of analyzing the data compared to other tasks: 4.8 out of 5.	
Test Coverage	The percentage of analysis of published tests done on average components	26/299 tests = 9%	Uncertainty and risk

Relevance of analysis	Percentage of tests analyzed that did contain errors	2/26 = 8% of analysis conducted did reveal an issue.	Increased cost

The researchers supplied information from interviews with data gathered from company databases. We base the number of tests on the average of the last 6 months. The portion of issues that are behavior related is 20.5%. We base this number on releases from the last 5 years.

The analysis process consists primarily of two steps. The first step is to locate the system parts at fault and the second step is to detect the causal factor. The time to locate the system part at fault had an average value of 15 hours a week (meaning ~2 hours per test) but the value varied between hours to weeks depending on the analysts. The researchers found the same varied results when asking the analysts to determine the time to detect the causal factor. We found the time to detect the causal factor to be between weeks and months.

Scheduling and communication. The research revealed some company-specific issues with the current processes. Other companies can experience some of these issues and are therefore included in this paper. At the very least, it provides some reflections that can be useful to keep in mind when designing or reviewing a company's integration activities.

We identified issues with the communication between the test engineers, system engineers and the analysts. First, the analysts lack information about the purpose and the goal of the tests. This is the biggest challenge experienced, according to 80 percent of the analysts. System information is essential to understand how the macro events will affect the system parts. Secondly, the analysts seem passive to the testers and system engineers because of the time they used to give feedback. Last, the research discovered that 33 percent of the analysts sometimes do not get information about tests that they should have analyzed.

For scheduling challenges, the analysts are required for analysis of system tests much later in the process than when they design and work on their parts. Analysts are working on other projects. While integration efforts are iterative, the integration activities can be years apart. This means analysts may be unavailable for analyzing the data when the need arises, delaying the feedback to engineers. Results show that the project is not always scheduling analysts prior to the planned integration activities. This increases the feedback time in testing.

A hypothesis held by the researchers was that analysts might neglect the importance of analysis. The findings show that this is not the case. Analysts want to prioritize the analysis work, but state that the time given is not sufficient.

Process improvements. This research has quantified the challenges, and potential gains with detecting emergent behavior in a system. The findings in this section have shown weaknesses with the process and the scheduling and communication. The scheduling and communication issue can be solved independently of the approach used and will therefore not be discussed any further. We will further explore the time constraint challenges discovered and introduce some potential concepts for solving these.

Anomaly Detection Approaches

The researchers explored different concepts for improvement. Figure 4 shows the specific phases where we can reduce unwanted emergent behavior. The company can make improvements in the design phase, during preparation for integration or in the integration phase.

In the design phase, the focus is reducing the unwanted emergent behavior that can appear. While efforts are made to reduce emergent behavior through design improvements (Cummings 2015, Neto 2016, Singh & Kogut 2017), the system is still likely to contain unwanted emergent behavior unless methods are fault-proof. Improvement in the design phase suffers from a lower detectability, difficulty of change management and feasibility of introducing it with existing methods. The detectability is lower with the simulation and modeling concept due to the knowledge being low in the design phase, as the learning process has not kicked in. Changes introduced post-design phase will need to be tracked and verified through the simulations and models, potentially introducing bigger changes. This research has not prioritized this concept because of a lack of verified methods currently existing in this area (Mittal et al. 2018).

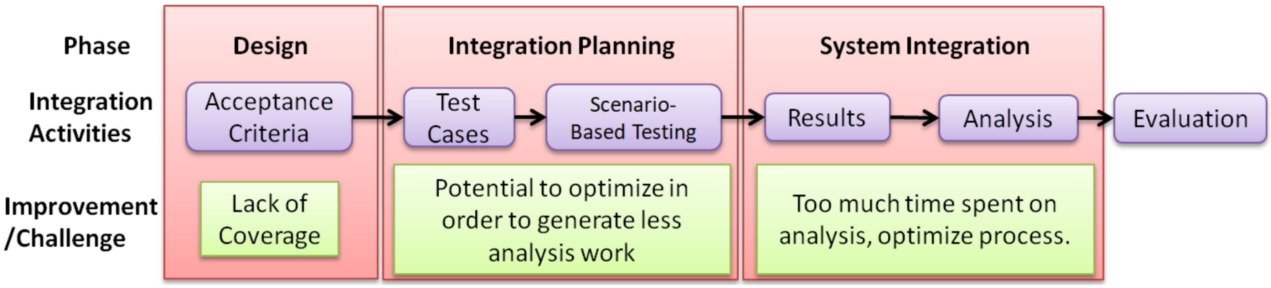

Figure 4. Challenges in different product development phases

Testing Approaches. The next step in the selection process is finding the correct concept to implement that fulfills the overall requirements the best. This research explores two approaches based on keeping track of behavioral changes for each integration. It also assumes development using an iterative development framework. The integrations can affect other parts of the system than first assumed, so having a way to measure and evaluate the effect of the behavior at system level can give a better way of tracking this.

Going forward, there are two potential routes looking at the amount of anomaly detection we want to introduce. One option is to focus on the macro-level. The solution proposed use system parameters. The accumulated value of the system parameters provides an indicator of the overall quality of the system. This is a less resource intensive option but yields little indication as to the system parts that cause the change of behavior in the system. The other option is to add anomaly detection for the micro-level. This can allow the anomaly detection algorithm the ability to identify what components are involved in causing the behavioral change. This is however costly in terms of initial cost and maintainability.

Macro-level. The approach focuses on picking a few system parameters that are likely to notice behavior changes in the system. An example of a system parameter, for a missile system, is the deviation from the planned route. The value for each system parameter is calculated and added together to give an indication of the system's behavior. The company can link this value to the specific test. When testers have conducted enough tests per increment, they can calculate an average value as the behavioral integrity indication. This value can indicate how well the behavior is working on the current integration. It will serve as a control mechanism to ensure expected behavior within defined limits when developers introduce changes. The company controls interface parameter values through test scenarios, and they expect negative tests that use out-of-bound parameter values to fail. If testers conduct enough tests per scenario, it can also give an indication of poor performance under certain situations, helping the developers to narrow down the potential causal factors.

Micro-level. Figure 5 shows the concept of using anomaly detection on the system parts. For each new development increment, the analysts should check each system part for changes in its behavior. A new increment is likely to change the systems behavior with intention, but it might change more system parts than those specifically intended. Figure 5 shows an example of this. The component in yellow has some changes introduced. This change affects the system parts in orange. The anomaly detection algorithm detects this, and the system part on the right was not expected. The system parts, where the analysis process detects behavior changes, need further analysis. The system parts that are not affected do not need further analysis. This means that analysts on system parts that are not affected can instead continue with the development. After an analysis on the relevant system parts, developers should make changes to the system parts if analysts have found unwanted emergent behavior. The company should approve the system parts if analysts have not found unwanted emergent behavior. The company will use the approved system parts as the new baseline for the algorithm to analyze behavior that the system assumes to find.

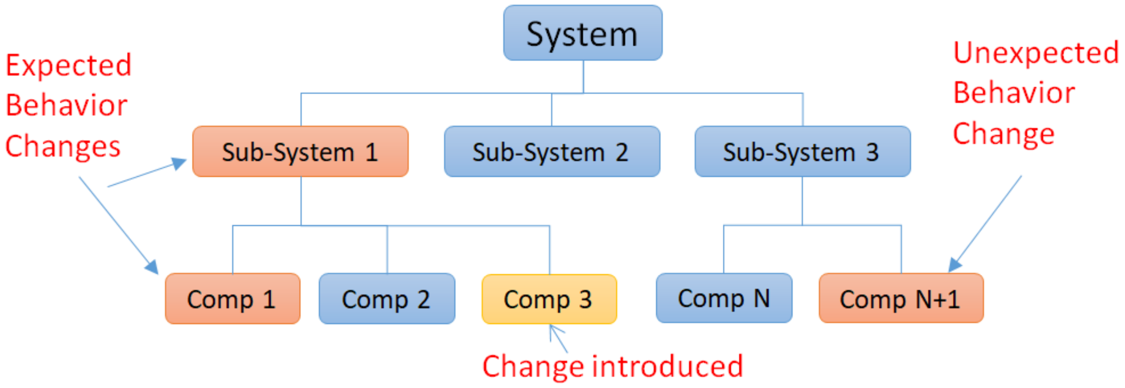

Figure 5. Anomaly Detection on System Parts

The advantages with the method proposed is that it can detect which individual system parts that are affected. This reduces the time required to find out what system parts cause specific errors. This is also likely to catch other types of errors that are not necessarily behavior related. Having a continuous approach will also aid with reducing the time for detecting errors, and speed up getting feedback to developers. Analysts conduct analysis of system parts in 9 percent of the tests. Reducing the time between analyses can reveal errors quicker resulting in faster feedback and potentially discover additional errors. The approach provides a control measure for behavior changes.

Discussion

This paper can be linked to the INCOSE IS 2021 theme “Accelerating through Adversity” by investigating the utilization of manual (human) vs automatic (machine) adversity to accelerate the detection of emergent behavior in system testing.

Findings. This work attempts to detect and quantify the challenges and constraints with detecting emergent behavior in missile systems. This section goes into detail of numbers and the rationale behind them. This section covers the approaches explored and thoughts on how well they can improve the current process.

- **What are the current challenges in detecting emergent behavior?**

The current challenges in detecting emergent behavior are unpredictability, scheduling, and communication issues.

Ill-suited process. A problem with well-defined, quantifiable acceptance criteria is that this method does not work well for capturing unforeseeable issues. The customer does not specify all the expectations to the product. Cultural differences can create problems of the extensiveness of the requirements because what seems intuitive for some might not be for others. Interactions between system parts can introduce system behaviors the developers have never experienced. This lack of knowledge results in unknown behavior when integrating the system parts. Unknown or unforeseen behavior that occurs due to interaction of system parts is difficult to formulate with foresight alone. This can cause a lack of tests checking that the behavior is as desired. If such tests have a weak coverage, it may lead to emergent behavior being difficult to catch during system integration. While it would be possible to make an extensive list and automate all test cases, it would be very time and resource demanding. Even with all that effort, unforeseen errors can still occur without having an explicit test case capturing it. The lack of ability to detect all emergent behavior through acceptance criteria has resulted in using manual analysis. The resources required for manual analysis is however, causing a bottleneck.

There are several communication challenges regarding the information presented according to 80% of analysts. In addition, some analysts also did not receive information about relevant tests. Analysts desire more information about the test execution and expectation to system level as well as better views and tools for analyzing the data. This poor communication leads to late detection. This is due to analysts spending longer time or not being included early enough in locating the root causes.

Better project management can solve the scheduling issues found during the research. The projects do not request analysts to perform analysis formally, but the analysts rather receive informal notification about the work required. This informal notification creates conflicts with other projects, which the project management can easily solve by requesting a certain workload from the analysts during the low-intensity months. A common practice within software development is incremental based testing and sprint periods with regular meetings for communication purposes among involved parties. This can potentially help the communication and scheduling issues seen at system level if the company forms work packages to exploit this opportunity. The project management can help by planning for analysis activities within different project increments, which again will lead to work packages planning for -and following-up analysis activities.

- **How are cost and resource constraints affecting the detection of emergent behavior?**

To supplement acceptance criteria manual analysis is used, this process is however resource demanding. The project deems the current workload as too large and that it is primarily required in certain project phases. This leads to non-continuous testing and poor test coverage causing slow feedback and risk of late detection. We can look at manual analysis in high-intensity periods and low-intensity periods separately. During the high intensity periods analysts use 39% of their time analyzing data from tests, covering most of the tests conducted. However, in low-intensity periods the analysts use only 4% of their time and only covers 9 percent of the tests conducted. Eight percent of the tests manually analyzed revealed errors. This number is high and indicates that projects can benefit from additional analysis. It can be noteworthy that this number has a low sample size, and more research is required. Reducing the time between analyses can reveal errors quicker resulting in faster feedback and potentially discover additional errors. The approach provides a control measure for behavior changes.

At first glance, it might seem feasible to have the analyst spend 39% of their time on manual analysis throughout the entire project. By doing this, the project would have continuous testing and cover the tests conducted. However, the added workload in coming years will make it increasingly difficult even with the addition of more analysts.

The analysts are usually experts in their fields and understand the components, interfaces, and subsystems the best. The experts are highly desired for developing the system, and it takes a long time for them to develop those skills. In such cases with manual analysis and extensive testing one of two things tend to happen, either the analyst neglects the data, or they neglect the development of the system. While it is possible to mitigate this problem by having more analysts and developers, it can be inefficient compared to automated techniques that can reduce the time needed for analysis and evaluation. Late detection and resulting late feedback to development can cause hidden errors for extended periods. Automation methods can reduce the workload for analysts by changing their focus from everything to managing the automatically alerted exceptions that the analysts need to investigate further (Haugen & Mansouri 2020).

- **How can the company improve detection of emergent behavior during the system integration phase?**

The increase in workload in coming years has made automated approaches more attractive.

Approaches. The approaches presented require iterative development and focuses on detecting anomalies to verify new integrations.

Macro approach. A thorough analysis within the company has revealed that this approach gives some of the same value as the manual testing process. In most cases, the changes in behavior will be visible for the testing crew. Testers detect changes in the behavior from a mix of test cases and system parameters by visual inspection. The mix of test cases and acceptance criteria is a selection of mission objectives. In the few cases where testers do not detect anomalies, the impact of the behavioral change is also limited. Once the analysis detects abnormal behavior, an analyst documents it and registers an issue for further analysis and potentially re-engineering. The main issue with the existing process lies with the slow detection process and not the ability to detect. Implementing system parameters that monitor this will not improve the time to discover the root cause. When the company introduce an automated test process, they should include automation of the emergent behavior detection performed by the test crew during test execution.

Micro approach. Another approach explored in this paper is the location of anomalies on a micro-level. Several challenges emerge with this framework. One of these challenges is how to keep the training data relevant for a system under development. The system under development will probably have its functionality changed several times during development. This means the training data captured will at several times be outdated. This requires the algorithms to request new data when the company has approved a new behavior, which would be best suited with an automated test process.

Another factor that the company needs to consider is if it is okay to accept the system parts that do not show any behavioral change. The algorithms developed are unlikely to detect many behavior changes in the system. These behavior changes can be spikes, different accelerations, and take different shapes within the expected threshold. Detecting all changes to the data will likely make it too sensitive and trigger even without changes in the actual system, but just due to the different scenarios having slightly different output. The limits need to widen due to the effect of the variation from human interactions. This fine-tuning adds a challenge to the system. This brings up the opposite problem, the threshold for detecting behavior changes is likely to be too wide, to avoid the previous mentioned challenge, and especially when considering the effect the different scenarios are going to have on the system parts. This means the algorithm cannot be too sensitive and might not reveal the desired behavior changes.

An advantage with this concept is the ability it gives to detect the system parts related to errors. However, a similar structure is also feasible to achieve by the use of test cases connected to the system parts they are likely to trigger. By mapping the triggered test cases to the system parts affected, one can pinpoint the likely sources.

Macro vs. Micro approach. The system parameter concept is less resource intensive but does not provide much value due to the company's current development process already assessing the system's overall performance. Therefore, detection on system parts is considered, which is a step up in terms of resources, but in return provides the ability to detect abnormal behavior on a micro-level. There are however few systems that will benefit from this approach due to the high cost, both in terms of implementation cost as well as the maintenance cost.

One option explored during this research is the use of an automated testing approach that focuses on requesting additional testing when the training data no longer matches the current behavior of the system. This approach could make it more suited for working throughout the system's development. The approaches explored are best suited for companies that integrate directly towards a platform that the customers are using. It can also be useful for companies that have extreme reliability requirements, such as sending things to space. For most system developers, this type of framework will have a high cost-to-usefulness factor. One reason for this is the low amount of emergent behavior occurring compared to other failures. Table 2 gives a short summary of the advantages and disadvantages with each concept.

Table 2: Overview of the Macro and Micro Approach

Advantage and Disadvantages	**Macro Approach**	**Micro Approach**
Detect Behavior Changes Between Integrations	X	X
Fast Detection	X	X
Detect the System Parts Responsible		X
High Maintenance Cost		X
Scalability	X	

During the research, we have learned to recognize the importance of good system parameters. Detecting changes in the behavior of parameters can provide next to no value if the company does not select them carefully. Selecting good parameters require help from system experts and an overall understanding of the system's behavior. The parameters should cover the essential aspects and deviations in the parameters, and should indicate that the behavior of the system has changed. An example is how the deviation from the planned route provides insights into the flight system's performance. Changes in this parameter can indicate an improvement or deterioration in the missile's capability.

In addition, it is important to consider other phases and not only detect post-system test. There can be significant improvements by reducing the amount of testing needed to verify the system. If resource constraints with manual analysis are a concern, then this can reduce the workload without adding more automated methods that require a maintenance cost.

During the research, the advantages of having testers look at real-time data became apparent. Inspection of the systems macro behavior has led to the detection of many emergent behaviors. A transition to automated testing can potentially miss this advantage. It is therefore worth considering supplementing an automated testing process with automated anomaly detection on system parameters, as proposed in the macro approach.

We base our findings in this paper on a case study in one defense company, making them only applicable to this company. On the other hand, many of the findings may be relevant also for other companies both in defense and in other areas, depending on where they are in the process of optimizing the detection of emergent behavior through different approaches in system testing.

Conclusion

This work identified the following root causes affecting the company's ability to detect emergent behavior:

- Acceptance criteria cannot detect many unwanted emergent behavior issues.

- Poor interdisciplinary dialog, likely due to silo-driven development, has led to scheduling and communication issues between testers, analysts, and system engineers.

Unfit. Acceptance criteria do not capture emergent behavior well due to its unpredictable nature. The project uses manual analysis as a replacement, but this causes a bottleneck. This bottleneck leads to non-continuous testing where analysts use on average 39 percent of their time analyzing during hectic months and only 4 percent of their time in non-hectic months. The company runs tests continuously throughout the project, but they primarily analyze the test results during the hectic periods. This results in a lack of test coverage, where the analysts only check 9 percent of executed tests manually. This is a problem since manual analysis detects issues that automated test cases are unsuited for, such as the case with emergent behavior. The resources required to execute manual analysis continuously is approximately 1575 hours per month. This is not feasible with the current resources available.

In addition, 8 percent of tests that underwent manual analysis revealed an issue. This is a significant number and is likely high due to manual testing processes detecting issues in real-time alerting test managers of potential issues. Test managers are more likely to request a manual analysis of tests that an automated test process reports as suspicious, increasing the amount of faults found per test.

Scheduling and communication. This research discovered four challenges with communication and scheduling that resulted in extending the time required for manual analysis. First, the lack of information about the purpose and execution of the tests was the key challenge according to 80 percent of the analysts. Second, 47 percent of analysts stated that poor presentation of test data and the lack of tools were a challenge in the current analysis process. Third, the manual distribution of test data has occasionally not notified 33 percent of analysts about tests that they should have analyzed. Last, 73 percent of analysts mentioned conflicts with other tasks as a contributing challenge. Further inquiries revealed that the analysts are working on other projects when they are required for analysis work. The project does not "book" the analysts for analysis, resulting in conflicts with planned tasks from other projects.

Approaches. The company can mitigate the challenges identified in three different phases of the project development. These three phases are the design phase, preparing for integration, and the system integration phase.

We have explored automated test frameworks that detect emergent behavior between integration increments. The frameworks can improve the time constraint challenge and yield the following advantages: control of behavior changes, fault localization, and continuous testing.

The first approach utilizes anomaly detection top-down (macro approach). It calculates and accumulates the macro aspects to give an indication of the system's behavioral integrity. This approach has a low cost, is scalable, and is the approach fitting best in our company case. The second approach is more costly, less scalable, but it aids with localization of root causes. The localization is not exclusive to behavior errors alone. However, the company can gain a similar advantage by connecting the system structure and the acceptance criteria.

Further Work

The company should explore other approaches that are not a part of the company's current processes. One such approach is detecting emergent behavior in the design phase through models and simulations. The company can mitigate unwanted emergent behavior in the system design phase by better improving the understanding of how system parts interact with each other at an early stage in the development process. This approach utilizes the principle of front-loading. A relevant method in such an approach is model-based testing.

An alternative approach can be to detect emergent behavior in the integration phase through system testing. The company can mitigate unwanted emergent behavior in the system integration phase through a better understanding of how system parts interact with each other in different scenarios. Such an approach can detect additional emergent behavior that is not detectable in the design phase. A promising method in such an approach is machine learning. Tests generate many log files (>500) with large amounts of data (<1GB per log file).

A third approach is to search for emergent behavior during the whole life cycle of the product using system simulations combined with real time data. The goal is to detect even more emergent behavior than the first two approaches. A relevant method in such an approach is digital twin. The company can use a virtual model of the product to test extensively the dynamics of the system with necessary variations to uncover unwanted emergent behavior.

The company should look further into quantification of emergent behaviors. How many types of emergent behavior lay inherent in the system? Estimate probabilities of the different types -and the cost of detecting them vs not detecting them.

References

Bedau, M.A. (1997) Weak emergence. Noûs, 31 (s11), 375–399.

Brown, D.S. and Goodrich, M.A. (2014) Limited bandwidth recognition of collective behaviors in bio-inspired swarms. Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, pp. 405–412.

Chan, W.K.V. (2011) Interaction metric of emergent behaviors in agent-based simulation. Proceedings of the Winter Simulation Conference, Phoenix, AZ, USA, pp. 357–368.

Checkland, P. (1999) Systems Thinking, Systems Practice: Includes a 30-Year Retrospective. pp. 28-111.

Cummings, M. (2015) Identifying and quantifying emergent behavior through system of systems modeling and simulation.

Engel, A. (2010) Verification, Validation, and Testing of Engineered Systems. pp. 463.

Fromm, J. (2005) Types and forms of emergence.

Haugen, R.A., and Mansouri, M. (2020). Applying Systems Thinking to Frame and Explore a Test System for Product Verification; a Case Study in Large Defence Projects. INCOSE International Symposium, 30: 78-93. https://doi.org/10.1002/j.2334-5837.2020.00709.x

Hofbauer, J et al. (2011) Cost and Time Overruns for Major Defense Acquisition Programs

Holland, J. (1999) Emergence, From Chaos to Order, Basic Books.

Holland, O.T. (2007) Taxonomy for the modeling and simulation of emergent behavior systems. Proceedings of the 2007 spring simulation multiconference, vol. 2, Society for Computer Simulation International, pp. 28–35.

Huttenlocher, D.P., Klanderman, G.A., and Rucklidge, W.J. (1993) Comparing images using the Hausdorff distance. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15, 850–863.

IEEE Automated Testing (2000) IEEE Aerospace and Electronic Systems Magazine, Jubilee Issue, October 2000.

Johnson, C.W. (2006) What are emergent properties and how do they affect the engineering of complex systems? Reliability Engineering and System Safety, 12, 1475–1481.

Kolawa, A and Huizinga, D (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society. pp. 74.

Kubik, A. (2003) Towards a formalization of emergence. Journal of Artificial Life, 9, 41–65.

Lineberger, R and Hussain, A. (2016) Program Management in Aerospace and Defense - Still Late and Over Budget

Mittal, S et al. (2018) Emergent Behavior in Complex Systems Engineering: A Modeling and Simulation Approach

Mogul, J.C. (2006) Emergent (mis)behavior vs. complex software systems. Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, New York, USA, pp. 293–304.

Potts, C. (1993) Software-Engineering Research Revisited. IEEE Software, Vol. 10(5), pp. 19-28.

Railey, B. (2019) Python Machine Learning - A Practical Beginner's Guide. pp. 46-65.

Rainey, L. B., and Tolk, A., eds. Modeling and Simulation Support for System of Systems Engineering Applications. Hoboken, New Jersey: John Wiley and Sons, 2015

Seth, A.K. (2008) Measuring emergence via nonlinear granger causality. Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pp. 545–553.

Shalizi, C.R. (2006) Methods and techniques of complex systems science: an overview, in Complex Systems Science in Biomedicine, Springer, New York, pp. 33–114.

Singh, S and Kogut, P. A. (2017) Detection and classification of emergent behaviors using multi-agent simulation framework (WIP). Society for Modeling and Simulation International (SCS).

Sols, A. (2014) Systems Engineering, Theory and Practice. pp. 42, 56, 219-230.

Szabo, C. and Teo, Y. (2012) An integrated approach for the validation of emergence in component-based simulation models. Proceedings of the Winter Simulation Conference, pp. 2739–2749.

Thyago, P et al. (2019) A systematic literature review of machine learning methods applied to predictive maintenance. Computers and Industrial Engineering 137 (2019) 106024.

Wolf, T. and Holvoet, T. (2004) Emergence versus self-organization: different concepts but promising when combined. Conference: Engineering Self-Organizing Systems, Methodologies and Applications.

Biography

Kent Aleksander Kjeldaas is a project engineer in Kongsberg Defence and Aerospace. He has 3 years of experience from research and development of missile systems, where he has worked within the Software and System Engineering field. He holds a bachelor’s degree in Computer Engineering from 2017. This report is the result of the research done for his master’s degree in Systems Engineering from the University of South-Eastern Norway in 2020.

Rune André Haugen is an industrial-PhD candidate at the University of South-Eastern Norway (USN). He was in service with the Royal Norwegian Air Force (RNoAF) from 1997 to 2003, including graduation from the RNoAF officer candidate school in Stavern (1999) and the RNoAF academy in Trondheim (2001). He holds both a bachelor’s degree (2006) and a master’s degree (2013) in Systems Engineering from USN. He has worked as a design engineer at FMC Kongsberg Subsea from 2006 to 2008 (3D modelling), and as a system engineer at Kongsberg Defence and Aerospace (KDA) since 2008 (system design and system test).

Elisabet Syverud is an Associate Professor in Systems Engineering at the University of South-Eastern Norway. She has 20 years of work experience from the industry, including three years at KDAs missile division as part of the Systems Engineering group.