Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Milosheska Bisera

Statična analiza izvorne kode v agilnih razvojnih metodologijah

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2016

University of Ljubljana
Faculty of Computer and Information Science

Milosheska Bisera

Static source code analysis in agile development methodologies

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: prof. dr. Viljan Mahnič

Ljubljana, 2016

Acknowledgments

I thank Mari Grini, Director of Security and Privacy, in Telenor Digital for believing in me and giving me the opportunity to collaborate with a renown team of world-class developers. Many thanks to dr. Tosin D. Oyetoyan, Post-doctoral Fellow in SINTEF ICT, Department of Software Engineering, Safety and Security, for the guidance, motivation, patience and support he so self-lessly offered with no hesitation. I also thank dr. Viljan Mahnič, the super-visor of this thesis at the University of Ljubljana.

And last but not least, I thank my family for their unconditional love and support during the process of writing the thesis. This work is their merit, but I owe them much more than a thesis to dedicate.

Milosheska Bisera, 2016

To my family.

"Do what is right. Not what is easy."

Contents

List of used acronmys

acronym	meaning
CAS	Center for Assured Software
CI	Continuous Integration
CWE	Common Weakness Enumeration
IDE	Integrated Development Environment
MOT	Metric Output Transformer
MVNO	Mobile Virtual Network Operator
NIST	National Institute of Standards and Technology
SAMATE	Software Assurance Metrics And Tool Evaluation
SAST	Static Application Security Testing
SAT	Static Analysis Tool
SCA	Source Code Analysis
SDLC	Software Development Lifecycle
SPQR	Software Project Quality Reporter

Povzetek

Naslov: Statična analiza izvorne kode v agilnih razvojnih metodologijah

To magistrsko delo raziskuje statično analizo kode za varnostno testiranje v praktičnem in agilnem kontekstu. Proučevali smo primer norveškega podjetja Telenor Digital.

Cilj dela je študija izzivov pri vpeljavi orodij za statično analizo kode, predvsem z vidika razvojnikov, ki razvijajo programsko opremo po agilni metodologiji. Poleg tega študija raziskuje tudi orodja za statično analizo kode ob uporabi kompleta orodij za varnostno testiranje (NIST Juliet Test Suite). Orodja smo primerjali glede na natančnost in glede na število pravilno klasificiranih pozitivnih primerov (true positive rate), ter število pravilno klasificiranih pozitivnih primerov, pri katerih ni hkrati tudi nepravilno klasificiran pozitiven primer v neranljivem delu kode (discrimination rate) v istem testnem primeru.

Na koncu smo v Telenorju izvedli naknadno evalvacijo vpeljanega orodja za statično analizo, katere rezultati so bolje izpostavili izzive pri vpeljavi orodij za statično analizo kode za namen varnostne revizije v agilnem razvoju. Zanimali so nas najpomembnejši dejavniki za vpeljavo določenega orodja in kakšne kompromise so ekipe pripravljene sprejeti ob uporabi tega orodja. Z namenom da bi podprli uporabo takih orodij, smo tudi proučili pomembne metrike za evalvacijo le teh.

Ključne besede

varnost v agilnih metodologij, statična analiza za varnostno testiranje, neodvisna evalvacija orodij za statično analizo

Abstract

Title: Static source code analysis in agile development methodologies

This study investigates static code analysis for security audit in an industrial and agile settings. The case study is Telenor Digital, located in Norway.

The study aims to understand the challenges for implementing a static code analysis tool from agile developers perspective. The study investigated static code analysis tools on a benchmark security test suite (NIST Juliet Test Suite) in order to make an informed decision by comparing the tools on the basis of their true positive rate and discrimination rate. Lastly, a post-evaluation of the implemented static analysis tool at Telenor was performed.

The results of this work shed more light on what are the challenges for implementing a static code analysis tool for security audit in an agile settings. The findings also identify the most important factors for adopting a particular tool, the trade-offs the teams are willing to make to adopt this kind of tool and the relevant metrics for tools evaluation in order to support adoption of such tools.

Keywords

security in agile methodologies, static analysis for security testing, independent evaluation of static analysis tools

Razširjeni povzetek

Agilni način razvoja programske opreme, ki ga spodbuja Manifest o agilnosti [1], je danes zelo razširjen med razvojnimi podjetji po celem svetu. Tak način razvoja se zavzema za razmislek in prilagoditve v rednih kratkih časovnih intervalih, kar dosežemo s postopnim in ponavljajočim razvojem. V nasprotju s tradicionalnim razvojem programske opreme, se pri agilnem razvoju življenski cikli razvoja, od zahtev in načrtovanja do implementacije in testiranja, ponavljajo v vsaki iteraciji razvoja. Tak pristop skrajša čas razvoja programske opreme, kar prinaša hitrejšo izdajo produktov na trg kot kadarkoli doslej.

Zaradi visoke konkurence se prodajalci programske opreme trudijo svojim strankam ponuditi boljše, stabilnejše in varnejše izdelke. Največji izziv pri agilnem razvoju, s katerim se podjetja soočajo, je pospešena hitrost razvoja, ki ovira zmožnost zagotovitve vseh potrebnih korakov v razvoju. Način, s katerim lahko izboljšamo kvaliteto, stabilnost in varnost produkta v agilnem načinu razvoja programske opreme, je integracija orodij za statično analizo v proces razvoja.

I Uvod in opis problema

Nekateri za agilno metodologijo trdijo, da je neskladna z varnostjo [2, 3], saj je tradicionalni proces varnostnega inženiringa zelo kompleksen in zahteva veliko dokumentacije, katere pa agilen pristop ne spodbuja. Obstajajo pa poskusi dodajanja varnostnih aspektov v agilen proces, ki se osredotočajo

na vpeljavo minimalnih varnostnih aktivnosti v vsako fazo razvoja. Tak primer je Microsoft SDL [4]. V fazi implementacije je uporaba orodij za statično analizo kode priljubljena [5], vendar pa lahko ta dodatna aktivnost povzroči motnje v razvojnikovem delovnem ritmu. Orodja za statično analizo lahko javljajo tudi lažne pozitivne primere, za katere pa je znano, da znatno povečajo razvojnikovo obremenitev.

II Obseg naloge

To magistrsko delo raziskuje statično analizo kode za varnostno testiranje v praktičnem in agilnem kontekstu. Proučevali smo primer norveškega podjetja Telenor Digital.

Cilj dela je študija izzivov pri vpeljavi orodij za statično analizo kode, predvsem z vidika razvojnikov, ki razvijajo programsko opremo po agilni metodologiji. Poleg tega študija raziskuje tudi orodja za statično analizo kode ob uporabi kompleta orodij za varnostno testiranje (NIST Juliet Test Suite). Orodja smo primerjali glede na natančnost in glede na število pravilno klasificiranih pozitivnih primerov (true positive rate), ter število pravilno klasificiranih pozitivnih primerov, pri katerih ni hkrati tudi nepravilno klasificiran pozitiven primer v neranljivem delu kode (discrimination rate) v istem testnem primeru.

Na koncu smo v Telenorju izvedli naknadno evalvacijo vpeljanega orodja za statično analizo, katere rezultati so bolje izpostavili izzive pri vpeljavi orodij za statično analizo kode za namen varnostne revizije v agilnem razvoju. Zanimali so nas najpomembnejši dejavniki za vpeljavo določenega orodja in kakšne kompromise so ekipe pripravljene sprejeti ob uporabi tega orodja. Z namenom da bi podprli uporabo takih orodij, smo tudi proučili pomembne metrike za evalvacijo le teh.

III Metodologija

V magistrskem delu smo uporabili kombinacijo različnih kvalitativnih in kvantitativnih raziskovalnih metod. Kvalitativno zbiranje podatkov smo uporabili za boljše razumevanje konteksta, v katerem naj bi orodje uporabili. Za pridobivanje kvalitativnih podatkov smo prek spleta izvedli vrsto intervjujev s posameznimi člani razvojne skupine. Kvantitativni pristop za zbiranje statističnih podatkov smo uporabili pri polavtomatski evalvaciji zmogljivosti orodij. Po vpeljavi izbranega orodja smo izvedli tudi anketo.

IV Sorodna dela

Ideja o uporabi statične analize kot del razvojnega procesa se je pojavila že pred več kot desetletjem. O tej ideji prvi razpravlja članek Chessa in McGrawa [6]. Predlagata avtomatizirano analizo varnosti izvorne kode z uporabo orodij za statično analizo. Članek proučuje pristope statične analize in orodij, ki so bila v tistem času primerna za tako delo. Beznosov [3] raziskuje področje klasifikacije metod in tehnik za zagotavljanje varnosti, pri čemer statično analizo uvršča v skupino metod za zagotavljanje varnosti, ki jih je mogoče delno avtomatizirati. Ugotavlja, da jih je mogoče uporabiti pri vsaki iteraciji agilnega razvoja brez dodatne denarne ali časovne obremenitve. Aggarwal [7] predlaga komplementarno integracijo statičnih in dinamičnih orodij za analizo ter izvaja meritve v nadzorovanem eksperimentalnem okolju. Članek se osredotoča samo na eno ranljivost, to je odkrivanje prekoračitve medpomnilnika (buffer overflow). Pristop bi lahko izboljšali z upoštevanjem širšega nabora ranljivosti.

S povečanjem zanimanja za računalniško varnost se je v zadnjih letih pojavilo tudi več raziskav na to temo. AlBreiki [8] je predstavil ogrodje za evalvacijo orodij za varnostno analizo, kot so orodja za analizo izvorne kode, in ocenil nekaj odprtokodnih orodij za statično analizo. Raschke [9] je nadgradil evalvacijski pristop z vpeljavo modela, ki se osredotoča na analizo

inkrementalnih sprememb v programski kodi. Smith [10] je uporabil nekoliko drugačen pristop. Osredotočil se je na iskanje informacij, ki bi razvojnikom olajšale iskanje in odpravo programskih napak, zaznanih s pomočjo statične analize.

Glavna slabost prej omenjenih pristopov je očitno pomanjkanje empiričnih dokazov iz podjetij, kar je poglavitna točka zanimanja v pristopu, ki ga predlagamo. Osnova za našo idejo je do neke mere izražena v članku Bace, v študiji primera, ki ocenjuje skupen življenjski cikel uporabe orodij za varnostno analizo glede na zmožnost odkrivanja ranljivosti, strategije za namestitev in uporabo orodij za varnostno analizo za identifikacijo ter odpravo ranljivosti. Članek prav tako navaja, da je »pristop, kjer je orodje integrirano kot del razvojnega procesa kot obvezni del, najboljša strategija prevzema, ki pa je učinkovita le, če so razvojniki ustrezno usposobljeni za uporabo orodja, kar jim omogoča odpravo identificiranih ranljivosti v čim krajšem času po njihovem odkritju«.

V naši študiji smo raziskovali vpeljavo orodij za varnostno analizo na praktičnem primeru ter ponudili empirične dokaze glede izzivov in metrik za vpeljavo takšnih orodij.

V Struktura naloge

V drugem poglavju (2) predstavimo pregled ozadja koncepta metodologije agilnega razvoja, izvor ideje in njene cilje, ter osnovna načela in prakse, ki opredeljujejo pristop agilnega razvoja. Nato opišemo dva primera takšnih metodologij: Scrum in Kanban. Poglavje sklenemo s pregledom koncepta varnosti pri pristopu agilnega razvoja programske opreme.

V tretjem poglavju (3) predstavimo koncept statične analize izvorne kode in splošne zmogljivosti orodij za statično analizo. Poglavje poudarja koristi uporabe orodij za statično analizo za detekcijo varnostno občutljivih programskih napak in predlaga ustrezno točko integracije v agilni proces razvoja. Da bi lahko naslovili prej omenjene pomanjkljivosti sorodnih raziskav,

smo skušali razumeti varnostne izzive v okolju agilnega razvoja programske opreme, identificirati omejitve orodij za statično analizo in ustrezne metrike uspešnosti vpeljave orodja za statično analizo v razvojnem okolju.

V četrtem poglavju (4) opišemo študijo primera, izvedeno v treh ločenih fazah: v fazi pred implementacijo (razdelek 4.2), v fazi neodvisne evalvacije orodij za statično analizo (razdelek 4.3), in v fazi po implementaciji (razdelek 4.4).

V petem poglavju (5) razpravljamo o ugotovitvah in prispevkih naloge ter o pomanjkljivostih pristopa in predlagamo ideje za izboljšave v nadaljnjih raziskavah.

Chapter 1

Introduction and problem formulation

The agile approach to software development, promoted by the Agile Manifesto [1], is a very widely established practice across companies worldwide. The approach advocates reflection and adjustments at regular short intervals which is accomplished by incremental and iterative development. In similar fashion, the regular software development life cycle (SDLC) practices covering the development phases from the requirements and design to the implementation and testing of a product, are also revisited during each iteration. In return, the approach shortens the time to market, making the race to product delivery move at a faster pace than ever before.

In this competitive scene, software vendors strive to offer better, more stable and secure products to their customers. The challenge that they are facing is the accelerating pace of development which hinders their ability to cover all the necessary steps in the development. One way of improving the quality, stability and security of a product in an agile development environment is to integrate additional static analysis tools in the deployment pipeline.

1.1 Problem statement

Agile methodology is argued to be incompatible with security [2, 3] since traditional security engineering process is very heavy and requires lot of documentation. There are attempts to develop agile compatible software security process. These attempts focus on introducing minimal software security activities at each development phase. For example, the Microsoft SDL [4]. At the implementation phase, using a static code analysis tool is shown to be popular [5]. Nevertheless, this additional activity may cause disturbances in a developer's regular flow. In addition, static analysis tool suffers from false positives which are known to significantly increase the developer's burden and effort.

1.2 Scope of the thesis

This study investigates static code analysis for security audit in an industrial and agile settings. The case study is Telenor Digital, located in Norway.

The study aims to understand the challenges for implementing a static code analysis tool from agile developers perspective. The study investigated static code analysis tools on a benchmark security test suite (NIST Juliet Test Suite) in order to make an informed decision by comparing the tools on the basis of their true positive rate and discrimination rate. Lastly, a post-evaluation of the implemented static analysis tool at Telenor was performed.

The results of this work shed more light on what are the challenges for implementing a static code analysis tool for security audit in an agile settings. The findings also identify the most important factors for adopting a particular tool, the trade-offs the teams are willing to make to adopt this kind of tool and the relevant metrics for tools evaluation in order to support adoption of such tools.

1.3 Methodology

A combination of various qualitative and quantitative research methods was used in the case study. Qualitative (exploratory) data collection was used to form a deeper understanding of the context which the tool had to be applied to. A round of individual interviews with the team members was conducted online. Furthermore, a quantitative approach was used to collect statistical data via a semi-automated evaluation of tools' capabilities. Lastly, a survey was performed after the chosen tool was implemented.

1.4 Related work

The idea of applying static analysis as part of the development process was introduced more than a decade ago and Chess and McGraw's paper [6] appears to be the first to discuss this idea. They propose automating source-code security analysis with static analysis tools. The publication provides an overview on static analysis approaches and tools that were at the time suitable for the purpose. Beznosov [3] works on classifying security assurance methods and techniques and places static analysis in the group of security assurance methods that can be semi-automated, explaining that they can be applied during each iteration without significantly burdening the budget or causing time overheads in agile projects. Aggarwal [7] suggests a complementary integration of both, static and dynamic analysis tools and performs measurements under a controlled experimental setting. The paper focuses only on one vulnerability, that is the detection of buffer overflow. Hereby, the approach may be improved by considering a wider range of vulnerabilities.

As interest in security significantly increases in recent years, research on this topic has also intensified. AlBreiki [8] introduces a framework for evaluating security analysis tools (SATs) such as source code analyzers, and evaluates a few open source static analysis tools. Raschke [9] upgrades the evaluation approach, by introducing a model that focuses on reviewing only

the increment of the software. Nevertheless, a somewhat different perspective on the matter is taken by Smith [10], focusing on identifying developers' information needs in order for them to be able to find and fix faults in the code, detected by means of static analysis.

The main downside in the aforementioned approaches is evidently, the lack of empirical evidence from companies, which is the main point of interest in the approach we are proposing. The basis for our idea is to some extent reflected in Baca's paper [11], a case study that evaluates the overall life cycle of an SAT usage, considering ability to detect vulnerabilities, strategies for deployment, and usage of SAT to identify and correct vulnerabilities. The paper also states that "a configuration management approach where the tool is integrated as part of the development process as a mandatory part is the best adoption strategy that is only efficient if developers are educated in order to make use of the tool to correct identified vulnerabilities as soon as possible after detection".

Nevertheless, the approach used in this study is to investigate the implementation of SAT in industrial setting and provide empirical evidence regarding challenges and metrics for adopting a SAT.

1.5 Thesis layout

First, in chapter 2, we give a background overview on the concept of agile development methodologies, the origin of the idea and its objectives, as well as the founding principles and practices defining the agile development approach. We further describe two examples of such methodologies: Scrum and Kanban. The chapter is concluded with an overview of the concept of security in the agile software development approach.

Chapter 3 discusses the concept of static source code analysis and the general capabilities of static analysis tools. It emphasizes the benefits of using static analysis tools as security fault detectors and suggests a point of integration in the agile deployment pipeline.

Further, in order to address the aforementioned shortcomings of the related research work so far, we tried to understand the security challenges in an agile software development environment, identify the limitations of static analysis tools and relevant metrics for adopting a static analysis tool in an industry setting. Chapter 4 describes the case study that was conducted in three separate phases: pre-implementation phase (section 4.2), independent evaluation of static analysis tools (section 4.3) and post-implementation evaluation (section 4.4).

Finally, chapter 5 discusses the findings and contributions of the thesis, as well as the shortcomings of the approach and proposes ideas on improvements for further research.

Chapter 2

Agile development methodologies

The agile software development approach has evolved as a direct consequence to the ever increasing needs for shortening a product's time-to-deliver. Customers expect to get ready-to-assess product functionalities in the early stage of a product development in order to be able to amend the initial requirements and adapt them in a way to achieve technical feasibility. Therefore, rapidly changing software requirements are driving the market dynamics into a state where variations of a product must be delivered often, in short time intervals, i.e. iterations. In response to this market dynamics the basic idea of the agile software development approach was born.

The agile software development approach superseded the traditional, "plan-driven" software development approach. The development process in the traditional approach is sequentially executed in five consecutive development phases: architecture, design, coding/debugging (implementation), quality assurance (verification) and release management [12]. In contrast to this approach, the agile approach propagates a development methodology where all of the development phases are intertwined, can be executed in parallel and are revised at each iteration. These practices yield the benefits of developing software iteratively and delivering increments of a product in the early

stages of development, instead of delivering a product as a whole in the final development stages.

Precisely its iterative nature is the greatest advantage of the agile development approach. This opportunistic development approach provides a possibility for customers to give early feedback and make sure that the development of the product itself moves in the right direction. It enables early detection of gaps between business expectations and developers understanding, discovery of customer needs, rather than customer wishes and early discovery of technical barriers [2]. Providing feedback which may lead to refinement of the initial requirements and specifications is crucial to the efficiency and productivity of the product development, as well as to the quality of the product itself.

The formal establishment of the main principles of agile methodologies roots back to February 2001, when a group of consultants met in Utah to share their experiences, practices and needs [13]. Based on their common opinions, they instituted a manifesto, "The Manifesto for Agile Software Development", to summarize the four important elements to value:

Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan

By imposing these directions to the developers community, the Agile Alliance set the foundations to a whole new software development mindset. Nevertheless, they do not disqualify the values on the right, they just prioritize the values on the left. These four basic elements have been widely accepted and valued ever since the establishment of the manifesto.

- **Individuals and interactions over process and tools**: The human factor is appreciated as the most influential factor in a project development to affect the quality of the product and success of development.

Formalization of processes is considered inflexible and therefore unsuited for an agile approach.

- **Working software over comprehensive documentation**: Piles of documentation did not provide any particular added value to customers and the focus is now reoriented to completed and working portions/-functionalities of a software in order to be able to provide frequent feedback. The amount of documentation produced is determined by the customers' own requirements.

- **Customer collaboration over contract negotiation**: Negotiating over a product's functionalitites, technical feasibility and overall customers requirements is now done at each iteration and is a part of the development cycle itself. This means that the contract no longer contains detailed requirements specifications, but the same purpose is accomplished through customer collaboration.

- **Responding to change over following a plan**: In contrast to the traditional development approach where a final plan is established at the early stages of a project initialization, the agile approach propagates progressively defining smaller goals, i.e. a goal per iteration. This enables progressive growth of the product, as the initial requirements are iteratively revised and amended.

2.1 Principles

The establishment of the manifesto was followed by the creation of the "Principles Behind the Agile Manifesto". It is consisted of 12 principles which present general truths that support the manifesto and serve as guidelines to the teams that implement the agile methodology:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.

4. Business people and developers must work together daily through the project.

5. Build projects around motivated individuals.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2.2 Practices

There are a number of activities that support the principles of the agile software development approach. These specific activities denote the application of the basic features offered by the agile development approach to a real project setting and are also known as the agile development practices. A team that intends to adopt the agile software development approach, must

consider implementing the set of practices that best fit its preferences and goals, as well as the nature of the project itself.

In continuation, we provide a more detailed description of the basic practices that portray an agile software development life cycle. In accordance with the main focus of this research work, we are particularly interested in the iterative and continuous nature of the agile software development approach along with the practices that define it, as well as the infiltration of the testing practices in a SDLC as defined by the agile software development approach.

- **Short Iterations**
  An **iteration** is a constant limited time interval, during which all of the development phases are covered. At the beginning of each iteration a narrowed scope of functionalities is defined which are covered by a set of user stories that describe the final goal of the iteration. The requirements and design are then revised in order to meet the new goal and are followed by implementation of the user stories. These activities lead to an iteration release which denotes the conclusion of an iteration. The product of an iteration is a subset of the general product and integrates in the final system, which evolves with each increment. The scope of an iteration is reduced to fit a regular predetermined timeframe. The usual length of an iteration is one-four weeks, but should be determined in an optimal way to suit the regular team's work organization and practices.

  Nonetheless, this particular agile development practice is what differentiates the agile approach from its predecessors in that, it enables customers to change their requirements even later in the process of product development.

- **Short Releases/Sprints**
  Customers are always kept in the loop with what is being developed, that is the features under development and the state of current develop-

ment. The customers get acquainted with the features that have been developed by frequent releases of the software, which enables them to test the software and provide feedback in the early stages of the product development. An increment of the software is usually released at the end of each **sprint**, which is consisted of a predetermined number of iterations. At the end of each iteration, the technical progress and challenges are discussed and on the basis of the customers' business experience and resources, they are allowed to change their business or development requirements and manoeuvre with the evolution of the product in the most favourable direction.

- **Code and Test**

  As working software has been advocated as one of the four founding elements of the agile software development approach by the "Manifesto for Agile Software Development", much attention is put precisely on the quality of the software. In order to be able to ensure better quality of the software they are developing, developers are now expected to include unit, acceptance and integration testing in the development process. This enforcement is intended to replace the need for documentation which gets outdated with time. This way, the testing can also be automated and the testing practices are part of the responsibilities of the development team. Automated testing ensures that the software being developed is always working properly and all changes made are backward compatible.

- **Continuous Integration**

  The automated testing is usually a part of the continuous integration practice, which is also a very valuable agile development practice. Usually, each developer develops on a local machine and pushes the code to a common develop branch at least once a day. After integrating the software, an automated build is performed which also runs the tests. If a test fails, the build will fail consequently and this will alert developers

to fix the bugs that have caused the failure.

- **Done Criteria**
  A successful integration build however, is not enough to ensure that a software is ready to be released. Sometimes, a team defines a list of conditions that need to be completed, before an iteration is considered to be "done". This criteria may vary from team to team and is determined by the team itself based on its own preferences. When all of the requirements specified in the "done criteria" are completed, then the software is ready to be released and delivered to the customer.

2.3 Examples of agile software development methodologies

Since the establishment of the Agile Principles, a number of agile software development methodologies have been been conceived. The methodologies emerge from the basic set of principles and practices, but propose a unique approach in embedding these practices into the regular development processes. The uniqueness of the approach is dictated by the priorities, preferences and perceptions of what a software development method should look like [14]. The following are amongst the most popular agile methodologies: Extreme Programming (XP), Scrum, Rational Unified Process (RUP), Lean Development (LD), Kanban, Agile Modeling, Feature-Driven Development (FDD) etc. Further, we will focus on presenting two of the aforementioned agile development approaches in greater detail.

- **Scrum**
  Scrum teams are expected to be an independent group of individuals who would be capable of functioning in a team where the organizational responsibilities are in complete control of the group itself. The team strives to achieve a common goal, which is defined by iteration. The

way in which the common goal will be achieved is completely entrusted to the skills of the team members.

As described in [15], Scrum is an iterative agile practice which supports incremental delivery of software. In the initial phase of the project, an architect designs the founding architecture of the system and defines the project's vision which is to be consistent throughout all the development phases. Developers must be able to make amendments to the initial architecture as requirements change during the project, which will be under the supervision and control of the leading architect. The requirements and architecture design are revised at each iteration, when developers specify a goal for a particular iteration and identify a set of tasks that will contribute to its achievement. Each developer is assigned a list of tasks that is feasible to complete during the iteration. The list of tasks that should be completed is called backlog and it is also revised at each iteration, as priorities and status of tasks changes. At the end of each sprint, which may be consisted of a number of iterations, potentially shippable software should be delivered to the customer.

- **Kanban**

The founding idea of the Kanban (eng. signboard) approach originates as early as in 1950s, when it was introduced as a manufacturing scheduling system [16]. This control flow mechanism advocates the principle of triggering processing activities by process demand signals.

Fifty years later, the same Kanban mindset was applied to software development. The idea was implemented by David J. Anderson, who was helping a team in Microsoft to improve its productivity. A key characteristic of the Kanban approach is the focus on prioritized features/functionalities at a given time, increasing productivity as developers focus only on particular set of tasks. This leads to limiting the work in progress and constantly releasing new working features. An-

other important characteristic of the approach is that it motivates team members to visualise the workflow and measure the cycle time. This provides greater transparency and motivates developers to constantly improve.

The Kanban approach is founded on the following principles [16]:

- Visualise the workflow
- Limit Work In Progress
- Measure and manage flow
- Make process policies explicit
- Improve collaboratively (using models and the scientific method)

2.4 Secure agile development

As elaborated earlier, the regular development flow in the traditional software development approaches spreads through the separate phases: architecture, design, implementation, quality assurance and release management. The phases in the regular SDLC are executed consecutively and security practices are embedded in each phase [12]. At first, **security functional requirements, abuse stories and misuse cases** are created during the requirements phase. Further, **threat modeling** is performed in the design phase in order to determine the possible threats. During the development phase, testing practices are performed, such as **static analysis and regression testing** and spread through the quality assurance phase where **dynamic testing** is performed. Lastly, **firewall policies and patches** to the deployment environment are performed during the release management phase as the concluding phase in the development cycle.

Nevertheless, we already know the agile software development practices good enough to be able to notice that the traditional security practices would not fit the agile deployment pipeline. According to [3], there are four conflict-

ing points at which traditional security practices confront the agile software development approach:

1. **Tacit Knowledge/Documentation:** A common practice in traditional security assurance is that developers and security evaluators are usually separated in order to stay independent and focused on their own work, as well as to be as objective as possible. Consequently, the practice induces the need for extensive documentation as a reimbursement for the knowledge not gained due to the dislocation. When it comes to the agile approach, the practice contradicts to the first of the four basic elements that the agile approach values. By prioritizing individuals and interactions over processes and tools, the approach advocates closer relations among coworkers, constraining the feasibility of the separation practice.

2. **Lifecycle:** The iterative repetition of the SDLC phases in the agile software development approach in contrast to the sequential SDLC phases in the traditional software development approach, affect the way the traditional security assurance practices are executed. Instead of performing a particular practice once at the end of a phase (threat modelling, regression testing, static analysis, dynamic analysis and firewall policies), the agile approach imposes the need of executing these practices iteratively. Consequently, this entails a need for automation in order to optimize the execution of repetitive security assurance practices.

3. **Refactoring:** The ability to respond to changing requirements is also one of the key agile development elements. When new functionalities are created it may happen that they oppose to the security constraints that are already in place. These constraints are set during the initial design phase, meaning that the agile approach again imposes the need of revisiting and amending these constraints at each iteration.

4. **Testing:** The focus of testing in the traditional software development approach is broad and covers the whole system. It is executed only once per SDLC and it is usually performed by a third party. In contrast to this, the agile software development approach entails a need to narrow the focus of testing down to the functionality level and implement it as an integral part of an iteration.

From the above discussions, it can be concluded that the crucial requirement imposed by the agile software development approach is to integrate the security assurance practices in the agile deployment pipeline and therefore, shift the responsibility to developers themselves. The code artifact goes through many sequence events during development, until deployment and delivery to the customer. It is processed by several tools which are integral components of almost every deployment pipeline and should be considered as potential points of security assurance integration:

- **Issue Tracker:** During the design phase, initial requirements are set and features are defined appropriately. The development of features is then fragmented to smaller tasks which are added to the backlog. Later, each developer is assigned a subset of tasks pulled out of the pool of tasks. Moreover, as features are developed and existing requirements change, additional tasks and bugs should also be recorded somewhere. It is very convenient to use a project management tool that keeps track of the issues and their status, which is exactly the basic point of the issue tracking tools such as Jira. It can provide an overview of the current project stage and the overall progress and pace of development.

- **Integrated Development Environment (IDE):** During the next, implementation phase, the developer starts working on the set of tasks that has been assigned to him/her by writing the code on his/her local machine. In order to make this experience more pleasant for the developer and to stimulate his productiveness, special development tools were created for that purpose. Such development tools are also known

as integrated development environments (IDEs) and the following IDEs are some of the most commonly used: Eclipse, IntelliJ IDEA, NetBeans, Microsoft Visual Studio etc. In addition to providing an editor for writing code, IDEs also provide a compiler or an interpreter for the code, a debugger and other helpful plugins to ease the programming, as part of the development workspace.

- **Version Control System/Source Code Management:** Coordinating with a team of developers and developing features on a same code base is not an easy task. A lot of problems may occur, from simultaneously working on a same file, overwriting another developer's code, to mixing up up-to-date code with older version of the code, or accidentally deleting some code/files etc., all resulting in code loss. Hereby, source code management software was developed to help teams coordinate their work and keep track of the code developed, along with more detailed information on who created it, who modified it and when did the changes occur, identifying points of conflict and alerting about them early enough, before they affect other parts of the code base. Version control systems or source code management tools such as Git and Subversion (SVN) handle checkout, branching, forking and other actions on the code repositories under their management.

- **Build Automation:** An agile software development approach requires automation at any point in the deployment pipeline. Routine actions performed on the code on daily or even hourly bases such as compiling the source code, packaging the binary code and running automated tests on the same code, can and most certainly should be automated. Thus, build automation tools have been created for the purpose of compiling, integrating and linking separate components of the code in the correct order, as defined by developers. Some of the most popular build automation tools are Ant, Gradle, Maven etc.

- **Continuous Integration (CI) Server:** While the aforenamed build

automation tools perform a set of tasks related to building the code locally, the same actions should also be triggered on the common code repository. As each developer commits the code which needs to be merged to the common repository, the code should be continuously integrated in a way that does not disturb the regular flow of the deployment pipeline. This is done by a continuous integration server, which detects updates committed to the code repository and automatically invokes compiling, building and testing actions on the updated code. Examples of such tools are Jenkins, Travis, Bamboo etc.

The potential points of security assurance integration we have identified, offer diverse opportunities for incorporating security assurance practices in the deployment pipeline. For instance, thread modelling may be done on sprint basis and security related bugs may be entered in the issue tracking system. Next, static source code analysis (SCA) can be infiltrated as part of a developer's IDE, a local automated build or a continuous integration server. Dynamic analysis can also be infiltrated in an automated build or executed by the CI server. Furthermore, build automation and a CI server offer opportunities for infiltrating unit, integration and regression testing as well.

In conclusion, the agile software development approach offers new and still inexhausted opportunities for integration of security assurance practices. To that end, we need to be inventive and innovative in our approaches to incorporating security as part of an agile deployment pipeline in order not to waste the unused potential. In that process, it is crucial to keep pushing the security practices as early in the deployment pipeline as possible, so that they are always up to date with prevailing requirements, it costs less to fix them and we learn early not to repeat them.

Chapter 3

Static source code analysis

Static source code analysis stands for code analysis that does not require an execution of a program in order to be performed. The initial static analysis practices consist of simple syntactic checks, performed by a grep or a find command [17]. Latter static analysis tools extended the very basic idea to capture the most common faults of a particular coding language and implemented it for various programming languages. The main purpose was to achieve automated source code review and in that way, mitigate the risks of code instability.

It is recommended that automated source code analysis is implemented as early in the software development lifecycle as possible in order to detect code faults in the early stages of development. Hence, when faults are detected earlier it is much less cumbersome for a developer to fix them while the number of faults is low. This will also prevent duplicating the faults, as developers learn the good coding practices in the early stages of the product development. It is also much less costly for the customer, as resources are used much more wisely. Moreover, the source code analysis is automated, so during its execution it does not require any effort from developers. However, SATs may also report false positives, i.e. faults detected by an SAT scanner that are either not real faults, or the team unanimously agrees that their presence is not harmful. Therefore, reports' review still do require developers'

assistance.

3.1 Capabilities

The capabilities of a particular static analysis tool may vary significantly across a spectrum of flaws that can be found by source code analysis. This type of analysis is performed on the source code in its original or compiled form, without executing the software. The analysis checks for faults based on a set of predefined rules which represent descriptions of the faults a tool aims to detect. However, the detection is not always completely precise and may be at times prone to detecting incorrect faults which may not be critical to the software execution, which are referred to as false positives.

Nevertheless, the main purpose of static analysis is to check for unsound code constructs, structural problems, common vulnerabilities and errors of the language itself, as well as control and data flows in a software. Hereby, static code analysis may provide better understanding of a system in the early stages of an SDLC. As further explained in [12], SATs generally scan for unhandled error conditions, unfiltered input variables, cross-site scripting, cross site request forgery, command injection, object availability and scoping, and potential buffer overflows. These issues are usually a direct consequence to programming errors.

Static code analysis may be used as an automated replacement for peer code review. Depending on the point in the deployment pipeline where static analysis is performed, developers will get reports on detected faults either in their development environment or as external reports. Either way, they will be able to act upon accordingly and learn from the findings very early in the SDLC.

3.2 SAT as a security fault detector

The inception of the idea to detect security vulnerabilities through source code analysis, as remembered [6], was a very modest variation of syntactic search for matches in a file, based on a set of predefined rules, which were supposed to reflect possible security vulnerabilities. According to Baca in [18], a failure is labeled as security vulnerable if, under any conditions or circumstances, it results in denial of service, unauthorized disclosure, unauthorized destruction of data, or unauthorized modification of data, which are all consequences of an exploit.

OWASP specifies a list of the most common types of security flaws that are often encountered in web applications, also known as the OWASP Top 10 list [19]:

A1-Injection: An injection occurs when untrusted data circumvents any input validation and tricks the system into executing malicious code or queries, which may even give the adversary access to the database, without authorization. SQL, OS, and LDAP injection are such examples.

A2-Broken Authentication and Session Management: Oftentimes, adversaries take advantage of weak implementation of the authentication and session management in an application and manage to steal sensitive data such as passwords and session tokes. The stolen information may be compromised and the identities may be abused.

A3-Cross-Site Scripting (XSS): Weak input validation may also allow adversaries to inject malicious scripts that trick the browser into executing them. This way attackers can take over user sessions or change the application's normal behaviour.

A4-Insecure Direct Object References: In a lack of appropriate protection mechanisms and access control, internal references to objects

such as files, directories and database key may be exposed to and result in data leakage when exploited by an attacker.

A5-Security Misconfiguration: Configuration settings are omnipresent throughout all layers of application development: application configuration, servers' configuration, platform configuration etc. Factory settings must never get to production, as attacker can easily detect and exploit such weaknesses. Therefore, secure configuration must be properly defined and always up to data.

A6-Sensitive Data Exposure: During development, a coder must be very careful about where and how sensitive information is stored and make sure that it is always validated before reaching the data access layer. One of the truly naive programming errors made is hard coding such data, as an adversary may be able to access it by the use of reverse engineering methods performed on source code binaries. Such data must be encrypted even when in transit.

A7-Missing Function Level Access Control: Lack of appropriate access control checks on the function level may enable adversaries to forge requests and gain access to functionalities bypassing authorization. Developers must therefore ensure that access control is properly handled throughout all parts of the application.

A8-Cross-Site Request Forgery (CSRF): A users browser is tricked into sending and HTTP request to a malicious web application, containing cookie information and other sensitive data about the user stored in the session, possibly even authentication data.

A9-Using Components with Known Vulnerabilities: Usually, vulnerabilities of public software, such as third-party libraries and frameworks are publicly disclosed and attackers may exploit these vulnerabilities to attack applications that use such components.

A10-Unvalidated Redirects and Forwards: As already mentioned, lack of validation may lead to serious exploits and force the software into unexpected behaviours. This also holds for validation of redirects and forwards, as it enables adversaries to redirect users to other malware pages or gain access to unauthorized pages.

To summarize, static code analysis does have the ability of identifying security vulnerabilities in source code and it is most effective if introduced early enough in the software development life cycle. It can provide early information about potential security issues present in the source code, prioritize them and suggest ways to fix them. Hence, if developers decide to act accordingly, they may even prevent the further spreading of the reported issues. Furthermore, the management and a security team may also be involved in the process and make decisions based on the static analysis reports.

3.3 SAT - point of integration in the agile deployment pipeline

Static source code analysis can be implemented either locally on a developer's workstation (IDE, build automation tool), or remotely (CI server).

- **On-premises:** an SAT can be used as IDE plugin or as part of the local automated build. Prior to committing the code to the code repository, a developer can run the static analysis in his/her own development environment and quickly navigate to the source of issue and apply the fix. There are multiple benefits of running static analysis on-premises: developers are given the freedom to customize the rules, can decide when to run the scanner and the code does not have to leave the environment.

- **Off-premises:** an SAT can be run as part of a remote automated build. The automated build is triggered on the code in the code repository and generated static analysis reports will contain information about

the overall changes comitted from all of the developers. The benefit of running static analysis off-premises is consistency of security rules and policies across teams.

Depending on their needs and preferences, a team is free to choose one of the aforenamed options, or a combination. The team also decides about the frequency of running static analysis. A common practice is to run it as part of the nightly build and immediately fix only critical issues. The list of issues should certainly be reviewed over the sprint meetings, where it should be decided which code changes are too critical to be released and the related issues should be added to the next sprint.

Chapter 4

Case study

Driven by the urge of the competitive market, developers are oftentimes required to deliver a product fast, in iterative short time intervals, inevitably affecting the short term quality of the product. However, a failure in security at any point of its development life cycle may lead to harmful consequences. In order to avoid this, security testing may be performed on a more frequent basis and security flaws may be discovered and addressed as early in the life cycle as possible, making it easier and less costly for the flaws to be fixed. Hence, we believe that integrating automated security testing in the development cycle itself may have the potential to foster greater level of security of the software developed.

Nevertheless, imposing additional activities may cause disturbances in the regular flow of the agile SDLC and subsequently, also affect developers' effectiveness. In order to understand the challenges that may arise as a consequence to implementing automated static code analysis in an agile software development environment, we have decided to perform a case study in an industry setting. In addition, the purpose of the case study is to also identify the limitations of static analysis tools and relevant metrics for adopting static analysis tool in an organization.

The case study is performed in collaboration with one of the teams in Telenor's research and development center, an incubator of modern and pro-

gressive approaches to digital communication - Telenor Digital. The case study was conducted on a software telco team, which is working on the implementation of a mobile virtual network operator (MVNO). Their network operator is built on top of Telenor Norway's radio network, which is accessed via an MVNO interface. The product is under development for somewhat more than two years at the moment of conducting the case study and it has still not been released for commercial use, but this is planned to happen in the near future.

It is the team's ambition to provide stable and secure mobile network services, that will meet the ever increasing security and privacy requirements and needs of end users. Being chased by the fast pace of product development, it is in developers best interest to detect and correct software faults as early in the development cycle as possible. Hereby, they will insure lower development time and fault quantity reduction in the long run. With that aim in mind, the team is considering to implement automatic static code analysis during development.

4.1 Methodology

In order to achieve the aforementioned goals of the case study, i.e. to understand the security challenges in an agile software development environment, identify the limitations of static analysis tools and relevant metrics for adopting a static analysis tool, the study was conducted in three separate phases: pre-implementation phase, independent evaluation of tools and post-implementation evaluation. The case study is used to address three main research questions and a separate research method is used to answer each of them.

- **RQ1:** What are the most important factors for adopting a particular tool?
- **RQ2:** Identify the SAST tools capabilities in order to support adoption by teams.

The pre-implementation phase was executed with the purpose to get an insight into the team's practices, regular flow of development and their expectations. The idea was to examine the context of an industry setting where the agile software development approach is implemented in practice and identify the security challenges in that context. In order to get an understanding of the developers' personal opinions, practices, wishes and expectations we have used qualitative research approach and performed individual interviews with 6/8 developers involved in the product development. The interview transcriptions can be found in appendix A.

Further, we suggest a set of static analysis tools that have the potential to yield improvements in the software security level. An independent evaluation of static analysis tools was performed in the second phase of the case study, aiming to identify the limitations of static analysis tools and to compare their performance. To derive these results, we used a quantitative research method and performed the comparison of the tools by running each of the scanners against a test suite designed specifically for comparing the capabilities of SATs in identifying coding faults which may lead to security vulnerabilities.

Lastly, a tool was chosen and implemented by the team during the third and last phase of the study conducted. The developer responsible for the implementation provided us with answers to the post-implementation questionnaire, which is available in appendix C. The answers helped us evaluate the team's satisfaction, future use intention, perceived usefulness and compatibility, as well as to identify the subjective norm.

4.2 Pre-implementation phase

As end users push the bounds of software expectations, software vendors often find themselves in competing roles to meet them. In order to understand the security challenges development teams are faced with, we have conducted a round of personal interviews with the developers in the team.

4.2.1 Interview design and questions

The interview is contained of five sections: professional background, personal opinion, agile practices, development environment and quality assurance (QA) and security. The first two sections were intended for all interviewees (all developers including the tech lead) and the last three sections were intended for the tech lead only. The data inquired in the last three sections provides and in-depth description of the context studied.

1. **Professional background:** The set of questions in this section inquire data on the developers professional background, such as job title, years of programming experience, any security related experiences and familiarity with security vulnerabilities.

2. **Personal opinion** (on system and security analysis): This section is intended to collect data related to the first research question, that is on developers expectations and challenges they fear with regards to static analysis tools implementation. Developers are asked about their current practices and their future expectations with regards to the use of static analysis tools during development. They answer on their fears, preferences and challenges they foresee with regards to integrating a static analysis tool in their regular development work flow.

3. **Agile practices:** The third section of the interview focuses on gathering data about the team's regular agile development practices. It collects data on which agile methodology is used, the frequency of software releases, developers' efficiency and velocity measurements, burndown charts, iteration length and iteration description in terms of the processes contained, pair programing and re-factoring practices.

4. **Development environment:** Gathering information on the team's development environment was important in making an informed decision when choosing the appropriate tool for implementation, as well as on choosing the point of integration in the agile deployment pipeline.

In this section, we collected information on the OS platforms, IDEs and programming languages used during development, their continuous integration practices and the existence of a testing, staging and production environment.

5. **Quality assurance (QA) and security:** The last set of questions was intended to gather information on the team's usual quality assurance and security practices. The questions focus on the testing/security testing tools in use, checks for security vulnerabilities, software quality measurement etc. In addition, the questions also inquire data on the security issues the team is faced with and would like to address, which is helpful in understanding and identifying the team's security concerns.

4.2.2 Context

Most of the information on the team's general practices were gained in the interview with the technical lead of the team. It gave us an insight into the current state of the software telco team, which helped us gain a better understanding of the context. The interview provides details on their regular coding and development practices, as well as on the usual work flow of the team, in general.

Agile practices

We found out that the team uses a combination of the agile methodologies Scrum and Kanban, which the tech lead referred to as "rainbow" approach to agile. At first, they were not too strict about this kind of team organization, so they started off using a Kanban-like methodology. Therefore, they did not follow any specific process framework, but they were simply working on a bunch of tasks and were doing the coordination in between. Now, they are moving towards more traditional Scrum-like development and they have created separate small groups of developers which specialize on separate

functionalities, such as business support system integration, hosting, setup and infrastructure. Coordination/sprint review is now done every two weeks, as each iteration/sprint lasts for two weeks. Coordination is done in order to make sure that they are all moving in the same direction, define the scope of the next sprint, i.e. the set of tasks that should be included in the next iteration and do a time estimation for those tasks.

An iteration in the Telco's case begins with scoping and defining relevant tasks for the following sprint. Then the developers work on the tasks and after a task is completed, a merge-request is done and the code is reviewed by a peer programmer. When accepted, the code is fit for being merged and it is pushed to the main branch. The code is then built on the build server and unit and acceptance tests are run. If a test fails, it will break the build. This process is continually repeated over each iteration.

In the past year they have invested their time and effort into setting up an infrastructure that will support a development process with much more frequent releases. Therefore, at the time of the interview it was difficult to specify the pace at which software releases were done. However, as most of the work on the infrastructure is now completed, they predict to release software every few weeks in the near future.

Despite their prior efforts to set up some metrics in order to measure a developer's efficiency, such as estimation of stories and measuring focus factors, the idea of measuring efficiency has turned out to be quite difficult to implement successfully in the team. Therefore, they do not use any techniques for measuring developers efficiency, as that does not conform with the team's mindset. They believe in team's efficiency, rather than individual's. And they also believe that the individual's feeling for its own productiveness is the most accurate measure for its efficiency.

Quality assurance

Quality assurance is performed in multiple phases actually. It starts in the design phase of the SDLC, when the team talks through the potential is-

sues. Next, Java being their main language, conformance to common coding standards for Java and additional documents proposed by Telenor Digital, is checked. Then code review is done, which is followed by unit and acceptance testing. Last but not the least, code is refactored all the time.

Despite all of these practices, the quality of the software is not measured. Also, no specific security testing is performed. Potential security threats are discussed during the design phase of the SDLC and bad practices are avoided while coding. Nevertheless, the team is aware of the existence of some design issues they have to fix, such as securing secrets and sensitive logs and because of that they would like to have automatic security checks on regular basis.

Development environment

Telco's developers are free to choose the development software they feel most comfortable with. Therefore, they develop on all three major OS platforms: OS X, Windows and Linux. They use various IDEs, such as IntelliJ, NetBeans, Emacs, Eclipse, Sublime... Their software is mostly written in Java, but they are also developing parts of it in JavaScript, shell script and Python. Jenkins is used as a build server for continuous integration. There is a separate testing, staging and production environment, which is about to be put in use in the near future.

4.2.3 Developers personal opinions

We have interviewed 6/8 software engineers, out of which 4 were on a senior level. The developers programming experience was varying from 4 to 37 years. Most of them did not have any particular experience with security-oriented tools and showed average familiarity with security vulnerabilities.

Professional background

Title	**Programming experience** (years)	**Experience with security-oriented tools**	**Familiarity with security vulnerabilities** (personal estimate on a 1-5 scale)
Software engineer	4	Does not recall	2
Senior software engineer	18	None	3
Senior software engineer	37	Not in particular	3
Senior software engineer	20	Has had experience with static analysis tools	3-4
Senior software engineer	20	Not in static analysis sense	3
Software engineer	6	None	4

Table 4.1: Developers professional background.

On the software

Regarding the safety criticality of the software, some of the developers didn't feel the software was safety critical at first thought. We then established that their first thoughts were related to the overall system stability and the consequences if it crashes. The system was also assessed as "not very safety critical" from infrastructure point of view, which would cover the tasks of handling firewalls and access rights to their servers. However, when the matter was considered from privacy and information security point of view, they all agreed that it is the most critical part of their software, due to the fact that they store sensitive personal data, such as personal conversations, messages and voice conversations. It may not be that peoples' lives are dependent on it, but leaking some of the highly sensitive information they

are storing might be quite dangerous.

On static analysis tools

The team has briefly used a static analysis tool called Sonar. They have implemented it as part of their build server – Jenkins, but it was disabled at the time the interviews took place, because their build server was broken. The tool was intended to run when their master branch was built by the build server and this is the point when issues are reported. The nature of the reports was not security focused. Alerts pointed to memory errors, referential loops, coding style, nonconformance to coding standard, declaration of variables, processing methods that are large or complex, test coverage checks... The tool was considered as too strict, as was mostly pointing to false positives and parts of the code had to be specifically marked in order to be skipped by the tool. However, they have acted upon the findings of the tool and removed some of the classical bugs from their application. Nevertheless, the infrastructure is ready and set to support the implementation of additional static analysis tool, unless it is decided to stick to the current plan, which is to continue using Sonar.

On challenges

At this point, we think it is very important that developers show awareness of the challenges they might have to face. The most important challenge is actually setting the tool up and getting it to work. Another challenge would be to convince the developers in the benefits of the tool in order to use it. Developers' greatest fear from implementing an additional tool into their regular SDLC are the invasiveness and time factors. Most of them are very optimistic about the positive effects that a static analysis tool will have on the quality of the code. However, they show concerns about time limitations, which might limit their ability to consider the tools alerts. They even proposed an idea to schedule a part of their working time for fixing security issues. Furthermore, they also fear that if the tool requires a lot of interaction

and reports misleading alerts, it might affect their productiveness and they may end up ignoring it. Yet another challenge they foresee is covering all of the programming languages they write code in, as well as the large volume of code base, which might cause great amounts of additional work on fixing existing flaws. Also, fixing some of the issues might also cause a conflict of different solution proposals. Apart from these challenges, they show awareness that developing with security in mind is harder, but necessary, as at current point they don't have any formal method nor formal software tools to address this and they rely on the general background knowledge of the developers.

The developers willingness to act upon the findings of the tool actually depends on the findings. As long as real vulnerabilities are in question and they are not getting huge reports of false positives, developers will be prepared to consider the alerts reported, some would even enjoy learning about and fixing the issues. However, they could not give a more confident statement about their willingness to act until they actually experience the tool personally.

On implementation

All of the developers agreed on the idea of implementing a static analysis tool both locally, as part of their IDE and remotely, as part of their build server. All, with the exception of one developer would prefer to have the tool implemented in Jenkins if they have to chose between the two approaches. They pointed out the benefits of automatic navigation and immediate feedback, when reports are rendered locally. They can promptly act upon the findings while they are still developing that particular part of the code. However, one of the developers finds this distracting, as these kind of alerts may interrupt him and get him out of the flow. On the other hand, having the tool implemented on the build server adds other benefits such as an overview on the performance metrics of the team as a whole, which they would not be able to have otherwise.

4.2.4 Conclusion

In summary, based on the information provided in the interviews we could conclude that the security of the software is mostly left in the hands of the developer. It is completely his/her responsibility and is dependent on his/her own knowledge. Nonetheless, the team is aware of the existence of some design issues and the need for automating the security testing process and they are strongly positive about it, so long as they have the funding to support it.

In order for the implementation and adoption of such a tool to be successful, it is of great importance that the tool requires the least effort and gives some benefit already in the short term, no matter how small. Therefore, the independent evaluation of static analysis tools should provide reassuring evidence that the hassle of implementing a particular tool is worthwhile. Moreover, we have to take great care in properly configuring the tool, so that it points out to the most relevant issues. We must focus on a set of vulnerabilities that will bring added value to the quality of the software. That is a learning process for the developers as well and it will take time to gain the knowledge to do it right.

4.3 Independent evaluation of SATs

In the second phase of the study we conducted an independent evaluation of static analysis tools as security analyzers. The focus of the evaluation was the tools' ability to discover potential security vulnerabilities, i.e. flaws that may eventually cause a failure that can be exploited. This kind of static code analysis is also known as static application security testing (SAST). In evaluating the tools, we followed the approach proposed by NSA's Center for Assured Software [20].

Firstly, we decided to conduct the evaluation on an existing, artificial software in order to simplify the process of evaluation and get more consistent results. Artificial software contains intentionally embedded vulnerabilities

and comes with the benefit of providing information on the exact or at least approximate location of the flaw. This way, it is easier to separate a true from a false positive. Therefore, we decided to use an artificial software developed specifically for the purpose of evaluating static analysis tools, that is the Juliet Test Suite for Java v1.2 [21].

Further, we use CAS's classification of Common Weakness Enumeration (CWE) entries to a spectrum of weakness classes. Each tool reports vulnerabilities in a unique format and each vulnerability is given a tool-specific message code. We mapped these codes to appropriate weaknesses defined in the Common Weakness Enumeration list. Each CWE detected by a tool was then classified in one of the following classes: authentication and access control, buffer handling, buffer overflow, code quality, dead code, control flow management, race condition, encryption and randomness, error handling, file handling, information leaks, initialization and shutdown, injection, miscellaneous, number handling and pointer and reference handling.

Lastly, the true positive (TP) rate and the discrimination rate for each weakness class, for each of the tools, following the approach used in [20]. The SATs were later evaluated on the basis of that data.

4.3.1 Juliet Test Suite

The National Institute of Standards and Technology (NIST) Software Assurance Reference Dataset (SARD) Project provides a collection of test suites intended for the purpose of evaluating security tools' performance on discovering security vulnerabilities in a source code [21]. A future goal of the dataset is to cover all phases in a software development life cycle (SDLC), from the initial concept, through design and implementation, to acceptance, deployment, and operation. Therefore, for the purposes of the independent evaluation of SATs performed in this project, we decided to use a test suite created by NSA's CAS. The Juliet Test Suite v1.2 for Java was chosen because it is specifically developed for assessing the capabilities of SATs, as this project particularly focuses on improving the implementation phase of the

SDLC automated static analysis integration in the deployment pipeline.

The test suite contains 25,477 test cases with 841 flaw types embedded in various control-flow and data-flow patterns. The complete Juliet Test Suite v1.2 for Java covers 112 CWEs, a subset of which covers 11 out of the 2011 CWE/SANS Top 25 list of security flaws [22]. The test cases represent a sample of artificial software that has these flaws embedded in the source code. Each test case targets one particular flaw which relates to a most relevant CWE entry, however, other flaws may also be unintentionally present. Therefore, each test case contains exactly one primary bad and one primary good method, which in addition may contain a call to another secondary bad/good construct consequently.

4.3.2 CWE/SANS Top 25

The MITRE Corporation is a non-for-profit organization and its main role is the operation of research and development centers. Under the sponsorship of this organization, a community-developed formal list of common software weakness types has been created, named Common Weakness Enumeration (CWE) [23]. CWE is a dictionary that provides information on existing software security flaws, which can help to identify and mitigate software security vulnerabilities. The dictionary can also be used as a baseline for comparison of security software that targets these flaws.

In cooperation with the SANS Institute, as well as other security experts, together they have created and maintain the CWE/SANS Top 25 list of most dangerous software errors, which uses the Common Weakness Scoring System (CWSS) to rank the errors [24]. The ranking is based on several factors, such as attack frequency, impact or consequences, prevalence, and ease of detection among others.

The table bellow shows the CWE/SANS Top 25 coverage and it also presents the mapping between the CWEs reported by CAS and the actual 2011 CWE/SANS Top 25 list of CWE entries. The last column represents the number of test cases in the Juliet Test Suite v1.2 that relate to a particular

CWE.

2011 CWE/SANS Top 25		**CAS Test Cases**	
Rank	**CWE Entry**	**CWE Entry / Entries**	**Java**
1	CWE-89:Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	CWE-89	2220
2	CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	CWE-78	444
4	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS), CWE-81: Improper Neutralization of Script in an Error Message Web Page, CWE-83: Improper Neutralization of Script in Attributes in a Web Page	1332
7	CWE-798: Use of Hard-coded Credentials	CWE-259: Use of Hard-coded Password, CWE-321: Use of Hard-coded Cryptographic Key	148
8	CWE-311: Missing Encryption of Sensitive Data	CWE-315: Plaintext Storage in a Cookie, CWE-319: Cleartext Transmission of Sensitive Information	407
13	CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	CWE-23: Relative Path Traversal, CWE-36: Absolute Path Traversal	888
19	CWE-327: Use of a Broken or Risky Cryptographic Algorithm	CWE-327	34
22	CWE-601: URL Redirection to Untrusted Site ('Open Redirect')	CWE-601	333
23	CWE-134: Uncontrolled Format String	CWE-134	666

24	CWE-190: Integer Overflow or Wraparound	CWE-190, CWE-191: Integer Underflow (Wrap or Wraparound)	4255
25	CWE-759: Use of a One-Way Hash without a Salt	CWE-759	17

Table 4.2: CWE/SANS Top 25 - Juliet test suite coverage.

The table bellow shows the rest of the CWE/SANS Top 25 list which is not covered in the Juliet Test Suite v1.2 for Java.

2011 CWE/SANS Top 25		CAS Test Cases	
Rank	CWE Entry	CWE Entry / Entries	Java
3	CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	None (Buffer Handling issues are covered in the related C/C++ test cases)	-
9	CWE-434: Unrestricted Upload of File with Dangerous Type	None (Design issue which does not fit into CAS Test Case structure)	-
10	CWE-807: Reliance on Untrusted Inputs in a Security Decision	None (Covered in the related C/C++ test cases)	-
11	CWE-250: Execution with Unnecessary Privileges	None (Design issue which does not fit into CAS Test Case structure)	-
12	CWE-352: Cross-Site Request Forgery (CSRF)	None (Design issue which does not fit into CAS Test Case structure)	-
14	CWE-494: Download of Code Without Integrity Check	None (Design issue which does not fit into CAS Test Case structure)	-
15	CWE-863: Incorrect Authorization	None (Design issue which does not fit into CAS Test Case structure)	-
16	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	None (Design issue which does not fit into CAS Test Case structure)	-

17	CWE-732: Incorrect Permission Assignment for Critical Resource	None (Design issue which does not fit into CAS Test Case structure)	-
18	CWE-676: Use of Potentially Dangerous Function	None (Covered in the related C/C++ test cases)	-
20	CWE-131: Incorrect Calculation of Buffer Size	None (Does not fit into CAS Test Case structure for Java)	-
21	CWE-307: Improper Restriction of Excessive Authentication Attempts	None (Design issue which does not fit into CAS Test Case structure)	-

Table 4.3: CWE/SANS Top 25 - entries not covered by the Juliet test suite.

4.3.3 Selected tools

In order to identify which SAT would best fit the needs in the given context, we have evaluated a set of tools that can be used for this intent in order to asses which of the tools will be most appropriate for the purpose of our further research. We used SAMATE's list of static source code scanners as a starting point in our selection process [25]. In addition, we also used other resources and information available on the Web. At last, we have made a selection of tools based on the language support, easiness of installation and configuration and appropriateness in terms of the process of comparison. We have selected five static analysis tools which focus on finding security flaws: FindBugs and its plug-in FindSecurityBugs, SonarQube, JLint, Lapse+ and a commercial static code analyzer.

FindBugs. FindBugs is a static source code analyzer which uses static analysis to look for flaws in Java software. It is an open source software, distributed under the terms of the Lesser GNU Public License. FindBugs analyzes the Java compiled classes, rather than the source code. The software is distributed as a stand-alone GUI application, but there are also plug-ins available for Eclipse, NetBeans, IntelliJ IDEA, Gradle, Hudson, Maven and

Jenkins.

Supported languages: Java.

Supported IDEs: Eclipse - versions: Mars (4.5), Luna (4.4), Kepler (4.3), Juno (4.2, 3.8), Previous to Juno (<=4.1), Neon (4.6).

Platform Support: Windows, Mac, Linux/GTK.

In order to narrow down the list of generated alerts to the focus of our interest, we executed the FindBugs standalone application and filtered only the 'bugs' found under the "SECURITY" category.

FindSecurityBugs. FindSecurityBugs is a FindBugs plug-in for security audits of Java web applications.

During the second approach, we added the FindSecurityBugs plug-in for FindBugs in Eclipse and we ran the plug-in, with the "SECURITY" category marked only and switched off all detectors in other categories. As expected, this approach yielded significantly more specific results, as the plug-in discovered much more security related flaws.

SonarQube. SonarQube software (previously called Sonar) is an open source solution performing quality analysis on source code. We have chosen the SonarQube Scanner for our analysis, as it provides a possibility to launch analysis from the command line.

Supported languages: ABAP, C/C++, C#, COBOL, CSS, Erlang, Flex/ActionScript, Groovy, Java, Java Properties, JavaScript, JSON, Objective-C, PHP, PL/I, PL/SQL, Puppet, Python, RPG, Swift, VB.NET, Visual Basic 6, Web, XML.

Platform Support: Windows, Mac, Linux/GTK.

Jlint. Similarly as FindBugs, Jlint also operates on Java bytecode. It performs a global control-flow and a local data flow analysis. Three main groups of messages are produced by Jlint: synchronization, inheritance and data flow. Jlint is distributed under the General GNU Public License version 2.0 (GPLv2). Binaries are available for the Windows platform and it must

be compiled form the source code for other platforms. However, we didn't manage to achieve this on OS X platform, so we ran the tool on a Windows platform.

Platform Support: Windows, Linux/GTK.

Lapse+. Lapse+ is an Eclipse plug-in which also performs static analysis of code with the purpose of detecting security flaws caused by the inadequate or non-existent validation of the user input data in Java EE Applications. It is an enhanced version of LAPSE, a software that was developed by the SUIF Compiler Group of Stanford University and LAPSE stands for Lightweight Analysis for Program Security in Eclipse. Lapse+ is distributed under the terms of the GNU General Public License version 3.0.

We have examined the first two out of the three approaches provided by Lapse+ to detect security flaws in a source code:

- Vulnerability Source: searches for points of code that can potentially be source of an attack.
- Vulnerability Sink: searches for points of code that can potentially be subjected to data injection.
- Provenance Tracker: performs backward propagation through the different assignations in order to determine if untrusted data may succeed in manipulating the behaviour of the website. When Vulnerability Source is reached from a Vulnerability Sink, it indicates a security vulnerability.

Supported languages: Java - version 1.6 or higher.

Supported IDEs: Eclipse Helios. Note: the standalone jar did not work for us on a OS X platform.

Commercial static code analyzer. The commercial static code analyzer performs static analysis on the source code for finding security flaws. It also provides explanations on the vulnerabilities, as well as remediation guidance.

Supported languages: ABAP/BSP, ActionScript/MXML (Flex), ASP.NET, VB.NET, C# (.NET), C/C++, Classic ASP (w/VBScript), COBOL, ColdFusion CFML, HTML, Java (including Android), JavaScript/AJAX, JSP, Objective-C, PHP, PL/SQL, Python, T-SQL, Ruby, Swift, Visual Basic, VBScript and XML.

Supported IDEs: Eclipse, IntelliJ Ultimate, IntelliJ Community Android Studio, IBM Rational Application Developer (RAD), IBM Rational Software Architect (RSA), Microsoft® Visual Studio.

4.3.4 Initial idea

At first, the idea was to run the tools against a benchmark tool [26] using the set of test cases provided by the Jiliet Test Suite v1.2 for Java [21] and compare their performance. The idea of this approach was to help us generate statistical data about the tools' capabilities in terms of true/false positives/negatives alerts rates and range of detectable security bugs. On the basis of this data we would later make an informed decision about which tool to choose for further examination.

However, we came to a conclusion that the approach did not quite fit our needs. The OWASP Benchmark Project offers a very powerful open test suite designed to evaluate the speed, coverage, and accuracy of automated static, dynamic and interactive application security testing tools. The 1.2 beta version of the test suite is an executable web application which contains around 3000 test cases, which are mapped to an appropriate CWE number for the vulnerabilities they present. Despite supporting these features, we found extending the base test suite with additional test cases a bit complicated and time consuming. The project is very well adapted for testing and comparing tools against the set of test cases it already contains itself, but we ran into difficulties adapting it to run with the additional test cases offered by the Juliet Test Suite v 1.2 for Java.

4.3.5 Our approach

Because of the difficulties we experienced, we decided to search for a more appropriate solution for our approach that would offer an automated scanning of the particular test suite and comparison of the generated reports. The model suggested by Wagner in [27] described an approach that addressed the exact same setbacks.

Error types. Each tool reports vulnerabilities in a unique format which makes the comparison of tools a somewhat cumbersome process. In order to be able to compare the flaw detection scanners, we have to fit various result forms into a uniform format. This is done by the Metric Output Transformer (MOT) tool developed as part of the project Software Project Quality Reporter (SPQR) [28]. The MOT tool transforms each report into a CSV file, where each line contains details about each detected flaw, such as: name of the scanner, abbreviation of the error reported by the scanner, name of the file and line number where the error was spotted, as well as an error message reported by the scanner. The errors reported in the CSV reports are then mapped to CWE numbers using a tool-specific CWE mapping files, which are available in appendix B. These mapping files serve the purpose of mapping the tool-specific message codes to the most appropriate CWE number where possible, or even to multiple CWE numbers in some cases.

Error locations. Where possible, the exact error locations of the flaws embedded in the test suite were identified and based on this information we could determine the number of reported flaws that matched the exact line. We refer to these matches as correct line matches. Otherwise, the range of lines where a flaw was intentionally embedded was determined by identifying the lines of the opening and closing brackets of bad code constructs.

Automated analysis and comparison. Another setback in performing comparison of software vulnerability detection tools is the cumbersome pro-

cess of running each tool separately and comparing their performance against a defined list of vulnerabilities. For this purpose, Wagner has developed a utility program [29] that automates the process by nailing it down to four simple steps:

1. Running a scan on the Juliet Test Suite to locate the lines of errors.
2. Running the static analysis scanners against the test suite.
3. Converting the reports of the scanners to a uniform format.
4. Comparing the results and generating an overall report to show the scanners' performance.

The initial idea was to run all of the tools from the command line and automate the whole process from the starting point of execution, through the generation of reports and transformation to a uniform format, CWE mapping of the flaws detected and finally, to the comparison of results. As neat as this idea sounds, it turned out to impose a lot of limitations on the selection of tools, as we were limited to only use tools that provided the possibility to be executed from the command line. We only managed to do this for two tools on the same OS X platform: FindBugs and SonarQube (sonar-runner) and Jlint on a different Windows platform. Nevertheless, we then considered the possibility to generate results for each tool separately and manually transform the results to the uniform CSV format expected by the comparison tool developed by Wagner. In addition to this, a separate file for each tool had to be created that would map the tool-specific error to the most appropriate CWE(s), which is expected as an input by the comparison tool (appendix B).

4.3.6 Weakness classes

Each tool uses different techniques for identifying programming flaws and targets diverse flaw types. In some cases, even tools that target the same

flaw type may be capable of finding different variants of the flaw. Hereby, a tool's specific capabilities vary significantly across different SATs. In order to perform a better evaluation of the tools, we decided to identify the strong and weak areas of each scanner. To that aim, the CWEs contained in the Juliet test suite were further segmented to the following specter of weakness classes, as suggested in [20]:

- Authentication and Access Control
- Buffer Handling
- Code Quality
- Control Flow Management
- Encryption and Randomness
- Error Handling
- File Handling
- Information Leaks
- Initialization and Shutdown
- Injection
- Malicious Logic
- Miscellaneous
- Number Handling
- Pointer and Reference Handling

4.3.7 Improvements

The utility program already mentioned, which is used for the purposes of the independent evaluation is called AnalyzeTool [29]. The program developed by Andreas Wagner, automates the process of analysis and comparison of static software security scanners. The program is written in Python and it is publicly available on Github. We used the code as basis for performing an independent evaluation of SATs. However, during the process of evaluation we discovered a few bugs in the code, which we amended. We also added some additional features to satisfy the needs of our goals. The extension of the tool is available at [30]. Generally, we made the following improvements:

- The tool did not perform recursive scanning of the files in the test suite and a subset of the code (contained in subfolders) was omitted from the reports.
- FindBugs CWE mappings were extended.
- The evaluation process was completely automated for SonarQube. Additional logic was added to the analyzer tool that transforms the reports in XML format from the original JSON format reported by the tool. Additionally, a SonarQube metafile was added to the MOT tool that transforms the report from XML format to the expected CSV format. SonarQube CWE mappings were also added.
- The evaluation process was semi-automated for FindSecurityBugs, as the reports had to be generated externally. A FindSecurityBugs metafile was added to the MOT tool that transforms the report from XML format to the expected CSV format. FindSecurityBugs CWE mappings were also added.
- The evaluation process was semi-automated for Lapse+, as the reports had to be generated externally. The reports were then manually fitted to the appropriate CSV format. Lapse+ CWE mappings were also added.

- The evaluation process was semi-automated for the commercial tool evaluated in the study, as the reports had to be generated externally. The reports were then manually fitted to the appropriate CSV format. CWE mappings for the commercial tool were also added.
- Missing mappings of CWEs to weakness classes were added.
- A bug was detected in the counting of the issues and was amended appropriately.
- We considered the labeling of "real positives" meaning "number of issues which are found in files where also Juliet-Testsuite Issues are documented" to be misleading and changed it to "issues reported in bad test cases".

4.3.8 Results

Environment setup

The following environment setup was used to perform an independent evaluation:

OS platforms: OS X, Windows 10 (virtual machine).

Virtualization software: VMware Fusion.

Prerequisites: Python, Java, ant.

IDE: PyCharm EDU, Eclipse Helios, Eclipse Mars.

Tools: Commercial tool (launched as a standalone application); FindSecurityBugs, Lapse+ (run as plug-ins); FindBugs, SonarQube, Jlint (launched from the command line).

Test suite: Juliet Test Suite v1.2 for Java.

Utility program for automated comparison: AnalyzeTool [29].

Data collected

On the basis of the information collected from the tool, the following data is computed for each tool:

- **correct line matches (CLM):** issues of correct CWE type found at a correct line,
- **different line matches (DLM):** issues of correct CWE type found at a different line,
- **range matches (RM):** issues of correct CWE type found within a defined range,
- **different type matches (DTM):** issues of false CWE type found at a correct line,
- **none matching:** issues which are not documented in the testsuite,
- **no CWE:** issues that are not mapped to a CWE,
- **issues reported in bad test cases (IBT):** number of issues which are found in files where also Juliet-Testsuite issues are documented,
- **other issues (OI):** issues found in files where no issues should be found.

Results classification

In general, when scanners are assessed, the following categories of reported errors are considered:

- **true positives (TP):** an existent flaw is correctly detected,
- **false positives (FP):** a non-existent flaw is detected,
- **true negatives (TN):** a non-existent flaw is correctly not detected,
- **false negatives (FN):** an existent flaw is not detected.

Metrics computed

Because of the limitations of the AnalyzeTool stated in subsection 4.3.7 and the lack of appropriate information in the data collected by the tool in order to compute the evaluation metrics, we did not feel safe to use the collected data for evaluation of the tools. Instead, a separate metric computation tool was developed by Tosin D. Oyetoyan to support the AnalyzeTool. Based on the CSV reports and the CWE mappings for each tool, the support tool collected the following data:

- **different line matches (DLM):** issues of correct CWE type found at a different line„
- **range matches (RM):** issues of correct CWE type found within a defined range,
- **discriminations (Disc.):** issues of correct CWE type found within a defined range, without incorrectly reporting the issue in a non-flawed code[20],
- **incidental flaws (IF):** flaws found in files which are not the target of the test case.

Based on the data collected, the following metrics were computed:

$$TPrate = \frac{\#RM}{\#RM + \#DLM} \qquad \textit{(true positive rate)} \tag{4.1}$$

$$DRate = \frac{\#Disc}{\#RM + \#DLM} \qquad \textit{(discrimination rate)} \quad [20] \tag{4.2}$$

4.3.9 Scanner results

Commercial tool

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	206	244	161	8213	45.78	35.78
Code Quality	26	129	20	2238	16.77	12.9
Control Flow Management	3	83	3	6597	3.48	3.48
Encryption and Randomness	12	147	6	17522	7.54	3.77
Error Handling	0	17	0	1187	0	0
File Handling	0	0	0	16512	0	0
Information Leaks	18	50	18	1734	26.47	26.47
Initialization and Shutdown	1	19	1	43481	5	5
Injection	1560	2662	695	160327	39.95	16.46
Malicious Logic	5	49	3	3306	9.26	5.55
Miscellaneous	8	12	8	996	40	40
Number Handling	0	0	0	94501	0	0
Pointer and Ref. Handling	55	216	26	3948	20.3	9.59

Table 4.4: Commercial tool results.

FindBugs

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	1	0	1	176	100	100
Code Quality	34	3	34	88	91.89	91.89
Control Flow Management	34	0	34	220	100	100
Encryption and Randomness	0	0	0	312	0	0
Error Handling	17	0	17	284	100	100
File Handling	34	1	34	438	97.14	97.14
Information Leaks	0	0	0	7	0	0
Initialization and Shutdown	19	1	19	6132	95	95
Injection	123	5	123	10521	96.09	96.09
Malicious Logic	0	0	0	135	0	0
Miscellaneous	34	0	34	2	100	100
Number Handling	0	0	0	3726	0	0
Pointer and Ref. Handling	130	89	45	203	59.36	20.55

Table 4.5: FindBugs results.

FindSecurityBugs

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	272	62	271	245	81.44	81.44
Code Quality	0	0	0	4	0	0
Control Flow Management	0	0	0	180	0	0
Encryption and Randomness	141	154	35	1632	47.8	11.86
Error Handling	0	0	0	282	0	0
File Handling	30	2	30	682	93.75	93.75
Information Leaks	0	0	0	147	0	0
Initialization and Shutdown	0	0	0	742	0	0
Injection	2465	669	2458	8906	78.65	78.43
Malicious Logic	16	0	16	293	100	100
Miscellaneous	0	0	0	2	0	0
Number Handling	0	0	0	1785	0	0

Table 4.6: FindSecurityBugs results.

SonarQube

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	1	0	1	3244	100	100
Code Quality	102	54	50	1886	65.38	32.05
Control Flow Management	28	1	28	4093	96.55	96.55
Encryption and Randomness	0	0	0	6153	0	0
Error Handling	14	0	14	833	100	100
File Handling	0	0	0	9842	0	0
Information Leaks	0	0	0	1018	0	0
Initialization and Shutdown	15	2	14	26627	88.24	82.35
Injection	311	673	133	91653	31.6	13.52
Malicious Logic	17	33	2	1498	34	4
Miscellaneous	8	0	8	585	100	100
Number Handling	0	0	0	59675	0	0
Pointer and Ref. Handling	49	36	40	3139	57.65	47.06

Table 4.7: SonarQube results.

Lapse+

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	0	0	0	245	0	0
Code Quality	0	0	0	49	0	0
Control Flow Management	0	0	0	528	0	0
Encryption and Randomness	0	0	0	100	0	0
Error Handling	0	0	0	2	0	0
File Handling	552	576	24	469	48.94	2.13
Information Leaks	0	0	0	360	0	0
Initialization and Shutdown	0	0	0	1970	0	0
Injection	1998	2160	84	11024	48.05	2.02
Malicious Logic	0	0	0	48	0	0
Miscellaneous	0	0	0	2	0	0
Number Handling	0	0	0	3977	0	0
Pointer and Ref. Handling	0	0	0	176	0	0

Table 4.8: Lapse+ results.

Jlint

Weakness Class	# RM	# DLM	# Disc.	# IF	TPrate (%)	DRate (%)
Authentication and Access Control	0	0	0	22	0	0
Code Quality	3	0	3	45	100	100
Control Flow Management	0	0	0	33	0	0
Encryption and Randomness	0	0	0	17	0	0
Error Handling	0	0	0	7	0	0
File Handling	0	0	0	79	0	0
Information Leaks	0	0	0	9	0	0
Initialization and Shutdown	0	0	0	3	0	0
Injection	0	0	0	87	0	0
Malicious Logic	0	0	0	63	0	0
Miscellaneous	17	0	17	22	100	100
Number Handling	0	0	0	3	0	0
Pointer and Ref. Handling	105	26	84	196	80.15	64.12

Table 4.9: Jlint results.

4.3.10 Conclusion

Based on the data computed on the true positive rate and the discrimination rate, we have enough support information to provide a reflection on the tools performances.The results calculated for each weakness class sepa-

rately, clearly show that the tools capabilities significantly vary across different weakness classes.

Although the overall results on the performance of the commercial tool are generally better than the general performance of the rest of the tools across the specter of weakness classes, there are certain weakness classes, such as "Error Handling" "File Handling" and "Number Handling" in which the tool showed weak performance. On the other hand, FindBugs performed significantly better in the "Error Handling" and "File Handling" weakness classes, but showed poor performance in the "Encryption and Randomness", "Information Leaks", "Malicious Logic" and "Number Handling" weakness classes.

These conclusions hold for the overall results calculated for the rest of the tools as well. This leads us to a conclusion that a team can either make an informed decision about the choice of a static analysis tool based on its specific needs, or the team can use a combination of various static analysis tools in order to cover the complete specter of weakness classes.

4.4 Post-implementation evaluation

During the process of implementation of a static analysis tool at a software vendor, it is expected that both technical and non-technical challenges will appear. Technical challenges may appear during installation and configuration. On a non technical note, it may also happen that developers' insufficient security awareness hinders the way to a smooth integration process. Moreover, the integration of an SAT in an agile software development lifecycle may also cause disruptions in the regular flow and therefore, cause additional challenges.

As can be concluded by the context analysis of the industry setting where the tool was implemented, no objective measurements on the team's efficiency are performed. Therefore, in order to measure how successful the implementation of the SAT was and what kind of effect it will have on the development

work flow, we had to take a different approach and decided to collect that information via a questionnaire, available in appendix C. The questionnaire was answered by the developer responsible for the implementation process. It covered four categories of our main interest, that are further elaborated into greater detail.

Future use intention: The developer expresses positive thoughts about the team's future intention of using the tool. He believes that the team will eventually start using the tool in the future and that they will dedicate a part of their sprint time intended especially in solving issues reported by the tool.

Perceived usefulness: He is however, not convinced in the tools' usefulness and potential to reduce the number of serious security defects and consequently, improve the security of their product. He is skeptical about the tools effectiveness in finding real vulnerabilities and expresses higher expectations from the tool. In his personal opinion, the perceived usefulness of the tool "mainly depends on the functional domain, since this tool is very much needed for financial domain for its PCI DSS certification, but not so relevant for other domains".

On a more positive note, he believes that advantages of using the tool outweigh the disadvantages and believes that the implementation of the tool will raise developers' security awareness and teach them good coding security practices. He disagrees that the tool will distract developers in their work, have negative effect on the team's atmosphere, confidence, focus and their ability to deliver.

Perceived compatibility: The developer disagrees that the tool is compatible with the way he or the team organize their work, but remains confident that it will be best practice to implement the tool both locally and on the build server.

In his personal words: "The tool fits perfectly with Waterfall SDLC, but is not perfect fit for Agile/Scrum SDLC. The concept of "version" in this tool is based on legacy development strategy to have the analysis run on main code branch. It does not fit perfectly with concept of feature branches (feature being developed on these branches) and quality gates, such as allow/reject merging of these feature branches with the main branch based on automated testes or relative statistics from static code analysis, which SonarQube had, e.g. allow to merge if increase in low priority issues, in feature branch compared to main branch, are less than 5%."

Subjective norm: The developer is clear about his subjective attitude and states that no one has influenced his opinion in any way.

Chapter 5

Conclusion

The use of automated static code analysis in agile software development is used for early detection of programming errors which may potentially lead to failures that can be later exploited as vulnerabilities. The use of automated SAST is intended to reduce development time and costs and improve the quality of the software in the long run. It is researchers' aim to push the boundaries of such testing forward and suggest possible ways of integrating SAST into the regular SDLC, in order to help development teams optimize the process for themselves.

This study elaborates diverse aspects on the idea of integrating automated static code analysis in agile software development. The study contributes to the researchers' aim by identifying the most important factors for adopting a particular tool by a team in an industrial setting. The case study provides evidence that show the importance of least effort requirements by the tool in order to achieve successful implementation. The tool must also be able to show benefits already in the short term, no matter how small, in order to persuade the team in its capabilities and motivate them for future use.

Furthermore, the work also identifies the SAST tools capabilities in order to make informed suggestions that meet developers' expectations and in that way, support adoption of such tools in general. This study shows that a combination of SAST tools should be implemented in order to cover the

entire specter of possible weaknesses. However, it is up to the team itself to choose the most adequate approach to SAST adoption, the one that best fits their specific needs.

Nonetheless, we also experienced some difficulties in the process of conducting the case study and ran into obstacles, which we believe may have hindered our findings. During the evaluation phase of static analysis tools, true positives not intentionally embedded in the test suite could not be distinguished from the rest of the incidental flaws, as a human expert would have to manually review the list of findings and identify a particular finding as a weakness that needs to be fixed. Otherwise, a finding which may actually be a true positive is considered as a false positive. Another limitation to yielding better results in the independent evaluation phase might have been the incomplete CWE mappings files for some of the classes, as not all vendors provide information on the tool's CWE mappings.

Further, due to lack of time during the implementation of the commercial tool, we did not manage to identify the real challenges of implementation, the effects on the regular development flow, or to assess the effect of the tool on the quality of the code in the long run. The questionnaire can only be considered as a conclusion and a reflection on what we have done, but cannot assess the developers' general opinion. This data would have been very beneficial and further research in that direction is strongly encouraged.

In general, the findings of this work shed more light on the challenges developers are faced with when implementing a static code analysis tool for security audit in an agile settings. The study also provides an assessment on the capabilities of different tools, their weak and strong areas, as important factors for adopting a particular tool. Moreover, the study explains the obstacles that make the process of evaluation rather challenging.

Finally, the work sets solid grounds for further research in this area. As this work did not manage to automate the entire evaluation process of SATs, that might be a potential idea for future research work that will further support the adoption of SATs for security audit in agile software development.

Bibliography

[1] Manifesto for Agile Software Development, http://www.agilemanifesto.org, accessed: 2016-03-29.

[2] L. Othmane, P. Angin, H. Weffers, B. Bhargava, Extending the Agile Development Process to Develop Acceptably Secure Software, IEEE Transactions on Dependable and Secure Computing 11 (2014) 497–509.

[3] K. Beznosov, P. Kruchten, Towards agile security assurance, in: Proceedings of the 2004 workshop on New security paradigms, NSPW '04, ACM, 2004, pp. 47–54.

[4] Microsoft, SDL for Agile, https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx, accessed: 2016-11-28.

[5] T. D. Oyetoyan, D. S. Cruzes, M. G. Jaatun, An Empirical Study on the Relationship between Software Security Skills, Usage and Training needs in Agile Settings, in: 2016 11th International Conference on Availability, Reliability, and Security, ARES 2016, Conference Publishing Services (CPS), 2016, pp. 548–555.

[6] B. Chess, G. McGraw, Static analysis for security, IEEE Security & Privacy 2 (2004) 76–79.

[7] A. Aggarwal, P. Jalote, Integrating Static and Dynamic Analysis for Detecting Vulnerabilities, in: 30th Annual International Computer Software and Applications Conference, Vol. 1 of COMPSAC '06, IEEE, 2006, pp. 343–350.

[8] H. H. AlBreiki, Q. H. Mahmoud, Evaluation of static analysis tools for software security, in: 2014 10th International Conference on Innovations in Information Technology (INNOVATIONS), IIT '14, IEEE, 2014, pp. 93–98.

[9] W. Raschke, M. Zilli, P. Baumgartner, J. Loinig, C. Steger, C. Kreiner, Supporting evolving security models for an agile security evaluation, in: 2014 IEEE 1st Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE), ESPRE '14, IEEE, 2014, pp. 31–36.

[10] J. Smith, B. Johnson, E. Murphy-Hill, C. B., H. R. Lipford, Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, ACM, 2015, pp. 248–259.

[11] D. Baca, B. Carlsson, K. Petersen, L. Lundberg, Improving software security with static automated code analysis in an industry setting, Software: Practice and Experience 43 (2012) 259–289.

[12] C. Pepper, Secure Agile Development, Tech. rep., Veracode (11 2014).

[13] L. Williams, Agile Software Development Methodologies and Practices, Elsevier Inc., Department of Computer Science, North Carolina State University, Raleigh, North Carolina, USA, 2010.

[14] O. Hazzan, Y. Dubinsky, Agile Software Engineering, Springer-Verlag London, 2009.

[15] L. Rising, N. S. Janoff, The Scrum Software Development Process for Small Teams, IEEE SOFTWARE 17 (2000) 26–32.

[16] M. Ahmad, J. Markkula, M. Ovio, Kanban in software development: A systematic literature review, in: 2013 39th Euromicro Conference Series on Software Engineering and Advanced Application, SEAA 2013, IEEE, 2013, pp. 9–16.

[17] D. Baca, Automated static code analysis - A tool for early vulnerability detection, Ph.D. thesis, Blekinge Institute of Technology (2009).

[18] D. Baca, B. Carlsson, K. Petersen, L. Lundberg, Evaluating the Cost Reduction of Static Code Analysis for Software Security, in: Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security, PLAS '08, ACM New York, NY, USA, 2008, pp. 79–88.

[19] OWASP, 2013 Top 10 List, `https://www.owasp.org/index.php/Top_10_2013-Top_10`, accessed: 2016-11-29.

[20] C. Pepper, On Analyzing Static Analysis Tools, Tech. rep., NSA (07 2011).

[21] NIST Test Suites, `https://samate.nist.gov/SARD/testsuite.php`, accessed: 2016-03-16.

[22] S. Institute, MITRE, CWE - 2011 CWE/SANS Top 25 Most Dangerous Software Errors, `http://cwe.mitre.org/top25/`, accessed: 2016-11-27.

[23] MITRE, CWE - Common Weakness Enumeration, `http://cwe.mitre.org/`, accessed: 2016-11-27.

[24] MITRE, CWE - Common Weakness Scoring System (CWSS), `http://cwe.mitre.org/cwss`, accessed: 2016-11-27.

[25] SAMATE, Source code security analyzers - SAMATE, `https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html`, accessed: 2016-03-16.

[26] OWASP Benchmark Project, `https://www.owasp.org/index.php/Benchmark`, accessed: 2016-03-16.

[27] A. Wagner, S. J., Using the Juliet Test Suite to compare Static Security Scanners, in: 2014 11th International Conference on Security and Cryptography (SECRYPT), IEEE, 2014, pp. 244–252.

[28] R. Ploesch, H. Gruber, G. Pomberger, M. Saft, S. Schiffer, Tool Support for Expert-Centred Code Assessments, in: 2008 1st International Conference on Software Testing, Verification, and Validation, IEEE, 2008, pp. 258–267.

[29] A. Wagner, GitHub - devandi/AnalyzeTool, `https://github.com/devandi/AnalyzeTool`, accessed: 2016-09-03.

[30] A. Wagner, B. Milosheska, T. D. Oyetoyan, GitHub - biseram/AnalyzeToolExtended, `https://github.com/biseram/AnalyzeToolExtended`, accessed: 2016-11-30.

[31] F. Roeser, Can Software Security be successfully implemented in Agile software development?

[32] J. Chóliz, J. Vilas, J. Moreira, Independent Security Testing on Agile Software Development: A Case Study in a Software Company, in: 2015 10th International Conference on Availability, Reliability and Security (ARES), ARES 2015, IEEE, 2015, pp. 522–531.

[33] F. Roeser, Can Software Security be successfully implemented in Agile software development? A systematic literature review, online, accessed: 2016-03-03 (2011).
URL `http://iucontent.iu.edu.sa/`

[34] P. Burnard, P. Gill, K. Stewart, E. Treasure, B. Chadwick, Analysing and presenting qualitative data, British Dental Journal 204 (2008) 429–432.

[35] T. Charest, N. Rodgers, Y. Wu, Comparison of Static Analysis Tools for Java Using the Juliet Test Suite, in: International Conference on Cyber Warfare and Security, 2016, pp. 431–437.

[36] H. K. Brar, J. K. Kaur, Static Analysis Tools for Security: A comparative Evaluation, International Journal of Advanced Research in Computer Science and Software Engineering 5 (2015) 1085–1089.

Appendix A

Pre-implementation phase: Interview transcriptions

AGENDA: Qualitative data collection

Professional background.

Personal view on system and security analysis.

INTERVIEW QUESTIONS

Professional background

1. What is your job title?
2. How many years of programming experience do you have?
3. Have you had any previous experience with security-oriented tools? If yes, please share some details (what kind of experience, which tools, when).
4. How familiar with security vulnerabilities are you? (1-5)

Personal opinion (on system and security analysis)

1. How safety-critical is the software you are developing? (1-5)
2. How do you feel about bringing software security into operations? Would you find it distracting or helpful?
3. Which static analysis tools do you already use?
4. Have you used them to find specific security defects (Weaknesses and vulnerabilities)?
5. Do you fear the impact of implementing additional security tools on the workflow of your team?
6. Are you prepared to act upon the findings of the security tool?
7. What challenges do you envisage/foresee?
8. Does the team have the skill to use the static analysis tool?
9. Which do you prefer – IDE integrated static analysis tool or report generated by static analysis tool after build? Why?

Agile practices

1. Which agile methodology does the team use?
2. How often do you release versions of the product?
3. How do you measure developers' efficiency?
4. How do you estimate the team's velocity per iteration?
5. Could you provide data/project burndown charts for the past year?
6. What is the usual time length of one iteration?
7. What sequential processes are contained in/describe each iteration?
8. Do they have specific, documented coding practices?

9. Do you practice pair programming?
10. Do you practice refactoring of the code?

Development environment

1. Which OS platform do you use?
2. Which IDE do you use?
3. Which programming language do you use?
4. Which continuous integration tool do you use?
5. Do you have a separate testing, staging and production environment?

Quality assurance (QA) and security

1. How do you measure the quality of the software?
2. What are your usual QA practices?
3. Which testing tools do you already use?
4. Which security testing tools do you already use?
5. At which point in the SDLC do you check for security vulnerabilities?
6. Are there any specific security vulnerabilities you would like to address? If yes, which ones would you point out?
7. In your opinion, what are the important factors necessary for adoption of static analysis tool for the team?

WHEN:	19 MAY 2016 at 11:00 AM
WHERE:	Online
ATTENDEES:	Developer 1, Bisera Milosheska

BM: What is your job title?

KM: Software engineer.

BM: How many years of programming experience do you have?

KM: About 4 years in this job, previously I was in a different function.

BM: Have you had any previous experience with security-oriented tools?

KM: Not that I know.

BM: How familiar with security vulnerabilities are you? Cross-site scripting, SQL Injection, do you know what they mean, how they can affect the software. . . ?

KM: We are well aware of it. It has not been main concern, it's not something that anybody thinks that much about when you do programming. But I am aware of the issues.

BM: Let's say on a scale from 1 to 5, how familiar are you with security vulnerabilities?

KM: 2 maybe.

BM: How safety-critical is the software you are developing? On a scale from 1 to 5, how would you estimate it?

KM: I don't really know what you mean by safety-critical, but it's a Telecom so I would say it is 5.

BM: It's 5, yes. Because you are probably working with a lot of personal data that can be exposed.

KM: Yes, personal data and also I am working mainly with telecom side and I have issues like network interconnection and correctness and that kind of things. I would say that it's important, I would say 5, yes.

BM: How do you feel about bringing software security into operations? Would you find it distracting or helpful?

KM: I think it depends on how invasive it is. But I find that it will be helpful. It all depends what we are talking about.

BM: In which way it is implemented, probably?

KM: If it requires more work, you can say, and enforces a more strict way of developing things.

BM: Which static analysis tools do you already use? Do you use any?

KM: We have briefly been using code analysis tool, Sonar. It is kind of a tool that basically analyses your source code and flags issues with the source code. (The Sonar tool for code analysis.)

BM: So do you fix these issues that are flagged? Do you use it regularly?

KM: Currently (i.e. Sonar) it is actually disabled, but it is part of our build setup. We develop software locally, on our laptops and we deploy it to a software repository, in our case is Github. And when we push to the "master" branch virtually a master branch on Github, we have this backend service called Jenkins, which will build software. And as part of that build Jenkins will invoke this Sonar backend build, it is a kind of analysis tool and it will report back potential issues with the code. As we are speaking, that part is disabled. But we have been using it and the plan is to use it again.

BM: Have you used SonarQube to find any specific security defects, vulnerabilities, weaknesses?

KM: It's mainly... You can find code that is not correct code, but test

coverage. It also point all memory errors or code test coverage checks. But I am not overall familiar...

BM: Have you sanitised any of those errors that the tool reported?

KM: It is list of issues that it reports and then go to and correct things. But often it points those things that I actually think are not real errors (but code that breaks with the coding standard enforced by the tool) - appear as they are, it also points on things like how you do declaration of things, or you declare variables and such. And it has a bit strict way of what it accepts and in some cases I do actually go into the code and mark the code as a code that should be skipped by the tool.

BM: But do you think that some of the reported vulnerabilities can be ignored? Because the tool has set up too strict rules?

KM: Yes, the majority of things it reports are useful.

BM: Do you fear the impact of implementing additional security tools on the workflow of the team?

KM: Not if it works the way that we are using this Sonar tool.

BM: So you would prefer to have it in the build server?

KM: Yes, that's correct.

BM: Are you prepared to act upon the findings of the security tool?

KM: Yes.

BM: I suppose you are, because you already said that you are doing that.

BM: What challenges do you foresee?

KM: At least from my experience with the Sonar tool is that it sometimes complains about issues that are not really issues. I would like to... In the case of Sonar there are ways of marking codes as codes that should not be evaluated by Sonar, but that this is supported by and

works with Sonar. What I would like to get from these tools is that they actually point out things that needs to be taken care of. And I actually think that Sonar actually does that, but it should not... It should be able to... (? at 10:44) because of the way we write the code and structure the code. We often use code constructs that code that may trigger alert or be flagged by the security tool, but if there are ways of marking source code as code that should not be handled by the tool then it's ok.

BM: So you would prefer if the tool only checks for certain parts of the code? That you can specify that.

KM: I will use the tool for all the code, but there are certain small parts which we must be able to escape. Mostly in the way we declare things. It is mainly more like not being bothered by... Because if you look at the report of the tool, you get total sum of things it finds. But it could be that most of those are minor things and it looks bad on the report. So we kind of need a way of filtering out things that they are not very valuable or are not of importance and need to be in the report.

BM: Do you think that the team has the skill to use the static analysis tool? It probably has because it already uses it.

KM: Yes.

BM: And my last question was if you would prefer an integrated static analysis tool or a report generated by a static analysis tool after build, but I think you already answered that, that you would prefer to have it in Jenkins.

KM: Yes, that's correct. After the build.

BM: That was the list of my questions. (Skipped part of conversation, maybe it should be included as it was relevant to the interview ques-

tions as well) Thank you very much, thank you for your time.

WHEN:	23 MAY 2016 at 09:00 AM
WHERE:	Online
ATTENDEES:	Developer 2, Bisera Milosheska

BM: What is your job title?

BN: I think it's senior software engineer probably. I used to work in Telenor before and after 5 years I found my title was consultant.

BM: Consultant :) for 5 years...

BN: I was not aware :) but I think it's senior software engineer.

BM: How many years of programming experience you have?

BN: Well... about 20 years. I started meddling with it in mid-nineties.

BM: So during these 5 years you were consultant, you were actually an engineer? :)

BN: Well I was a programmer I was considering myself a programmer. That was my first job at Telenor.

BM: Have you had any previous experience with security-oriented tools?

BN: Security oriented tools, such as... give an example?

BM: Such as... I'm investigating SonarQube and FindSecurityBugs at the moment.

BN: In that case yes. I had experience with static analysis tools, I also had experience with firewalls. So I guess the answer would be yes.

BM: How familiar with security vulnerabilities are you?

BN: Well moderately... the article I was reading just 10 minutes ago was about just that.

BM: On a scale from 1 to 5, you would say 5?

BN: No, I wouldn't say 5. That takes a lot. And I've never been specifically interested in security it's more like something you really have to take into consideration. So, I would say 3-4 for probably.

BM: But you do have a pretty high awareness of security?

BN: Yes, but it's not my main interest.

BM: How safety-critical is the software you are developing?

BN: In this project it is not very safety critical.

BM: But you are working in the team?

BN: Yeah, but my part's not very crit... well actually in this project I'm mostly not working on software development, I am working on infrastructure automation. So maybe 20% of my job or 30% is actual programming and the rest is more architectural I guess.

BM: So it's not too safety-critical?

BN: Actually that part is extremely safety critical but in a slightly different way. So it wouldn't be safety critical like the case that would be bothered by an SQL injection or things like that, instead you would have to consider firewalls and access rights.

BM: Yes but it's not to safety critical in terms of what I want to investigate... in terms of the source code itself.

BN: So we should probably only consider the coding part.

BM: Yes. Do all of the developers I am interviewing work on separate projects?

BN: No. I am working with Kjell and Thomas also, so I'm currently at two projects and last week I was mostly working on this project which is a development one, creating a message store and assembly facility or modifying what we had and that has some security considerations.

BM: You are actually working on the same project but you have different roles.

BN: Yes.

BM: How do you feel about bringing software security into operations? Would you find it distracting or helpful?

BN: Could you please elaborate a little bit on that?

BM: Like implementing a software security tool such as SonarQube or FindSecurityBugs that would do a static source code analysis and report on the vulnerabilities that it has found. That would happen during the development process actually.

BN: That sounds wonderful.

BM: So wouldn't you find it distracting? Do you think that it is more helpful than distracting?

BN: It's absolutely more helpful. Developing software is perhaps more an art than science. It's not like when you are building a bridge, you don't have any formulas to guide you and there is a very real possibility that your software will contain a lot of errors, as opposed to the bridge which will not. So software engineering is not really engineering in my view.

BM: So any alerts (triggered) during the process will be helpful?

BN: I think so, absolutely. I'm not sure everybody would feel that way but I definitely like this. I have set up SonarQube and Cobertura and other static analysis tools previously when working with Java and I find it useful. And I also find continuous integration quite useful.

BM: Well I would like to hear your personal view, that's why I have personal interviews with all of the developers. I'm really glad you think that way.

BM: Which static analysis tools do you already use in this project?

BN: Bjørn Remseth set up SonarQube some time ago. Unfortunately, I don't think it is effective as our build server is currently broken, so I

think the answer would be that right now we don't use any.

BM: Why don't you think it's effective? Is it because no one is...?

BN: Because our build server is not building.

BM: No one is responsible for that tool maybe?

BN: It's complicated. It's part of a rather big thing, we want to have better testing processes and we want to have proper smoke tests, or acceptance tests, but doing that requires us to be able to automatically deploy recent versions of the software and that is something we attempted and failed, so we need to do that in a different way. So basically we need a way to have an environment where we can easily deploy more or less automated things – artefacts [the package or executable resulting from a software build]. We have that now, so the next step would be to actually do this automated deployment. We also need to implement versioning of the software, so that we know what version is actually running there. Unfortunately, everything is currently versioned 1.0, everything! Which I find a bit annoying to say the least, so that's something we need to fix. And then we can start having a build server, build automatically and deploy automatically and at that point, as a side effect of that we will get static analysis tool. So I guess you could say that static analysis is not prioritised, because we could probably just have it at the build server.

BM: Do you think that it will be hard to implement it for the needs of my project?

BN: No, not at all. Actually it only hasn't been prioritised, because the main priority has been running the smoke tests. That we developed, we have the smoke tests ready, just didn't manage to get the environment done. It's actually quite easy to simply set up the build server to just build the project and ship it off to static analysis.

BM: Yes, the implementation itself is trivial, I think.

BN: Yes, I think most of the setup is there. It's just never been, at least

not mine, priority.

BM: Have you used SonarQube in terms of the project for discovering some kind of vulnerabilities until now?

BN: I think there is no specific focus on security there. We had a few runs of SonarQube on our projects and the quality there is not very good. It reports huge amounts of issues.

BM: Of none important alerts?

BN: Some high, and some... most low I guess. I didn't look too closely, I just shook my head.

BM: Do you fear the impact of implementing additional security tools on the workflow of your team?

BN: No.

BM: Do you think that maybe if you get this kind of huge reports and you would have to act upon these reports, would you find it distracting maybe?

BN: No, I like learning (teaching?) [probably meant "learning about", but probably more in the sense that I like to have (lack of) quality quantified] this kind of things. If that was my primary job, to just fix issues, I would enjoy it probably.

BM: So you are prepared to act upon the findings of the tool?

BN: I wouldn't mind at all. I guess, maybe the issue is that sometimes is not just a simple fix, it's like OK, thing is just a crock of s*** and it needs to be rewritten and it's a huge job. That's sometimes that maybe a problem. And also, it could be I guess conflict in views, on how to solve issues. I think a lot of the things that SonarQube reports are more like code formatting issues. I don't know if you have worked with programmers before, you probably know that if you start arguing about how to format your source code and you could have a three hour discussion, because not everybody agrees. I mean this is

like someone likes the variable names like that...

BM: Yes, everyone has different kind of practices.

BN: Yeah. So many of the things that SonarQube reports are more on the aesthetic side and they always raise discussion, so you probably need to have coding style guidelines, not everybody agrees with them or you can do some of these fixes. That's not related to security. Security things are more concrete and needs to deal with.

BM: What challenges do you foresee? You answered a part of this question.

BN: When cleaning up/handling the SonarQube reports?

BM: Yes, when the tool is already implemented and the reports are there, do you think that...?

BN: OK, so we have a little bit of experience, as I said. Bjørn Remseth set it up half a year ago, I guess. And I had a look at it and I guess some others had a look at it and the problem is that when you set it up at this stage of the project we have a huge design debt, because I guess things were implemented quickly, rushed before summer last year and way to start. And of course in itself is not productive, nobody gives you a hug after fixing SonarQube reports, there's no new features, no performance improvements, in general. Maybe it's not seen as productive, in a sense.

BM: So you think that the programmers are not very aware of the improvements that this tool might make on the long run?

BN: Personally I think, and I am very fond of this term design debt. You create some piece of software, usually you have a deadline, you have to cut some corners and you ship it and at that point you will have some design debt and you will have to spend some hours every month just to write off that design debt and there is usually more features and most of them work when a software project comes after the initial release. That's almost always the case. And most of that work is

writing off the design debt. So I think that such a tool can be very good for managing the design debt and knowing where to work [what to work on]. Of course, design debt is not something you see directly, it is something that is only visible indirectly, because at some point, adding features becomes very hard because you are up to your ears design debt and it's hard to... it's inflexible, it's fragile, so on... But, back to your previous question, I think it is easier if you have a new project to start using such a tool. Because initially you have zero issues.

BN: Yes, but the idea is actually to implement it in an existing project.

BM: And then you have 2647 issues.

BN: Yes, but you can still adapt the tool, you can exclude some of the warnings if you want, if you think that they are not important.

BM: Yes, you can look at the critical ones. I guess you could maybe say that 'OK, Monday before lunch that's only fixing SonarQube issues. You can do things like that I guess.

BN: Maybe you can reserve a part of your working time just for that, just on security issues reported.

BM: Does the team have the skills to use the static analysis tool?

BN: I don't think that would be a problem. It's just like a glorified report, at least SonarQube. But, there are other things. I've only briefly looked at SonarQube now, but a few years ago I worked on setting up static analysis tools for Java and I recall that there was actually quite a lot of configuration, you need to customise the reports (? at 17:47). This kind of thing is OK, don't warn me on this, this is how we do it here. There is actually quite a lot of functionality in these tools that they don't necessarily have to deal with.

BM: Yes, definitely. We will focus only on a set of vulnerabilities, not everything that the tool is able to report. Because it will be too much

and won't do any quality research with all of what the tool can discover.

BM: I have one last question. Which would you prefer, an IDE integrated static analysis tool or a report generated by a static analysis tool after build? Would you prefer to have it as part of Jenkins?

BN: Yes, I understand. I think I have tried both and I don't think they are necessarily mutually exclusive, I think you can have both. But I find the one that comes after the build more useful because then you can't do very... First of all, it's authoritative across all developers, it's produced by a single tool, it's always like a single report and it contains the customisations of the team, not your personal customisations. It is not interfering with your development process, it's a bit like autocorrect in word processors. Kind of like 'Oh, is that mistyped? What was the correct spelling?' and suddenly you go out of the flow. I don't like to be interrupted. And you can also fail the build if the static analysis tool has a critical warning or a major warning. On the other hand, I think integrated in your local development environment it is nice sometimes, you can check for instance before committing, it's useful, but it's not as important, it is very nice to have. So the auto integration is mandatory, but local integration is just nice to have.

BM: Thank you very much for your answers. That was it. (Skipped conversation, irrelevant to the interview questions)

WHEN:	23 MAY 2016 at 10:30 AM
WHERE:	Online
ATTENDEES:	Developer 3, Bisera Milosheska

BM: What is your job title?

BK: Senior software engineer.

BM: How many years of programming experience do you have?

BK: 20.

BM: Have you had any previous experience with security oriented tools?

BK: I can't think of specific tools... Not in static analysis sense or... I guess, no... Not that I can think of. Maybe if you give some example, perhaps...

BM: In this project we are currently investigating static analysis tools, SonarQube and FindSecurityBugs are such examples.

BK: I haven't used those, no.

BM: How familiar with security vulnerabilities are you? On a scale from 1 to 5, how would you estimate your... (familiarity)?

BK: On a scale of 1 to 5, 1 being not very?

BM: Yes.

BK: 2 or 3, maybe 3.

BM: How safety critical is the software you are developing?

BK: It's not safety critical... it's not very safety critical, 1 or 2...

BM: Which project are you working on at the moment? You are all working in the same team and probably on the same project, but on different parts of it, right?

BK: That's true. But when you say safety critical that makes me think of airplanes.

BM: Oh, well... I mean in terms of security vulnerabilities and how could those affect the end users. Do you use a lot of personal data maybe that could be leaked?

BK: Ok, then in that case it is quite high because yes, we store their conversations, their messages and their voice conversations. So, if you mean with respect to data, then yes.

BM: I mean if a vulnerability could be exploited in any way, what kind of effect would that have?

BK: Potentially, it could have very negative effect for us. If someone got into our systems and then had access to our users both personal data and private conversations. That would have a very bad effect, both for the users and us, yes.

BM: So you would say that you store very sensitive data?

BK: Yeah, we store sensitive personal data, private messages.

BM: How do you feel about bringing security software into operations?

BK: How do I feel?

BM: Would you find it distracting or helpful?

BK: I would find it helpful, definitely. We don't have formal method or we don't have formal software tools yet to help in this. I think we rely on the general background knowledge of the developers. I think we've had a few presentations on the subject. Which have been helpful. We have had a workshop with a security company and that was useful, but that was more to get an awareness of typical vulnerabilities.

BM: But, would you find it distracting if there was another tool that would try to help developers with the discovering of some of the vulnerabilities?

BK: I wouldn't find that distracting, no. I'd find that useful.

BM: But maybe it would report a lot of vulnerabilities that aren't actually vulnerabilities, maybe it will have a lot of false positives...

BK: If it had a lot of false positives, then yes, people would start ignoring the tool. So, if the tool is tuned to work properly or can be configured...

BM: Yes, it has to be properly and thoughtfully configured.

BM: Which static analysis tools do you already use? Do you use any?

BK: I think we use Sonar for some of the Java code. We have a lot of JavaScript code, but I am not sure if we use Sonar for that. We might have used it once, but we don't use it regularly.

BM: Have you used it to find any security specific weaknesses?

BK: We haven't used it specifically for that. We use Facebook's infrastructure for writing, the Javascript framework that we use for our Web frontend has some support for warning you when you are vulnerable to XSS attacks for instance.

BM: What kind of framework do you use?

BK: React, framework for writing web apps. They help you a little by making it hard to create XSS scripting attacks in your code. You have to explicitly say this bit of code is unsafe in order to introduce that vulnerability as it were. But that's the only explicit example I can think of.

BM: Do you fear the impact of implementing additional security tools on the workflow of your team?

BK: I fear that a little bit, yes. I think it would be reassuring if we actually got something that the developers thought it was worthwhile. If developers are implementing something that they don't think gives the benefit then it would be an issue, but if they can see that it is helping then it shouldn't be a problem.

BM: Are you prepared to act upon the findings of the security tool?

BK: Yes, given what I just said. If I believe the finding and if I believe that that is a vulnerability and it will help, then yes. I guess that comes both to the tool and our understanding.

BM: What challenges do you foresee?

BK: In adopting the tool?

BM: Yes, in implementing it.

BK: We have several software languages that we write in. Predominantly Java and Javascript. But also some C++ as well. So to target each of those different languages would be an issue. And... I guess the volume of... We have quite a large set of software now. So it could bring up many many changes I suppose, a lot of work to do... Because the project has been running for two years.

BM: Is the code base too big to start with this kind of testing now?

BK: Yes, potentially the code base is too large and then it depends a bit on the nature of the changes that are required.

BM: Does the team have the skills to use the static analysis tool?

BK: Hopefully yes, we have used static analysis tools, so it is not totally unfamiliar. Maybe not with a security focus.

BM: Which one would you prefer? An IDE integrated static analysis tool or a report generated by a static analysis tool after build?

BK: I would probably prefer the IDE, but I would actually like both. My preference would be for the IDE.

BM: Which IDE do you use?

BK: I use a lot of SublimeText and plugins for SublimeText. And occasionally... recently that's the only one I've been using.

BM: I think that in the case of your team (the context the tool will be applied to), it will be more appropriate to use Jenkins integrated reporting because you are all using different IDEs.

BK: Yes, that's true. We are using different ones, so Jenkins would be a good fit.

BM: Up until now, you are the first developer that said that would prefer an IDE plugin.

BK: OK, I guess I was just imagining... As I said, both would be ideal, but...

BM: Would you find it very distracting if the build failed on Jenkins when the tool alerts about some high security vulnerabilities?

BK: Potentially yes, I am familiar with the problem of too much noise from these tools, because it's difficult to filter out what's relevant and what's not, as I mentioned. So potentially yes, but I haven't seen a tool that achieves that yet. But you generally configure it right, you switch off certain warnings in certain circumstances, because you know better than the tool. Yes, it might be distracting at first, but that's why I mentioned it should be possible to configure things, switch of the noise.

BM: Thank you very much. That was the set of questions, I am done with it. (Skipped conversation, irrelevant to the interview questions)

BK: Thank you, I hope that was useful.

BM: Yes! Thank you for your time!

WHEN:	24 MAY 2016 at 11:00 AM
WHERE:	Online
ATTENDEES:	Developer 4 (tech lead), Bisera Milosheska

BM: What is your job title?

BR: Senior software engineer.

BM: How many years of programming experience do you have?

BR: 37.

BM: Have you had any previous experience with security-oriented tools?

BR: Not particularly, no.

BM: How familiar with security vulnerabilities are you? Let's say on a scale from 1 to 5?

BR: 3. I mean I am aware of it, I just have never worked/focused on it.

BM: Yes, yes. But you are aware of what kind of security vulnerabilities exist and what they mean.

BR: Yes I am.

BM: How safety-critical is the software you are developing?

BR: I would say it's pretty significant.

BM: If you were to estimate it the same way as the previous question, on a scale from 1 to 5, what number would you choose?

BR: 3-4. It's probably not going to be people's life depending on it on a regular bases. But it could actually be. So it's not in the same class as as say nuclear weapons or in general weapons technology or life-support technology or things of that nature. But it is communication services and we do have somewhat sensitive and in some cases even

highly sensitive information. So releasing it could be quite dangerous. So I would say that it's pretty important, but it's not the most important thing in the world. But I mean that quite literally, I mean nuclear weapons would probably be the most important thing and...

BM: Yes, I was expecting that kind of answer, because all of the developers almost said the same. That you are working with pretty sensitive data, with personal data and that that's the reason why you would indicate that it is pretty much safety critical.

BM: How do you feel about bringing software security into operations?

BR: I am not sure I understand the question.

BM: If a tool would be implemented as is the intention now with our project, would you find it distracting or helpful?

BR: That of course depends on how the tool goes into the workflow. In general, we do use automatic tools for many things. One of the things that they are in general very good at is just picking out known and easily formulate-able issues that we need to fix, so type errors, stylistic errors in the code, when integration fails and things of that nature. Things that we... that is known to us and that we do want to have support in the software. And something difficult issues every now and then, run this tests and we do it all the time. If you can get it on that level then it will be pretty nice. Because it will make us more aware of the practices and enforce it in consistent manner and this is kind of workflow that developers are usually quite used to and comfortable with. If on the other hand the tools require also interaction or doing things that are not very productive and actually not really contributing to security , because it is just window dressing, then it wouldn't be very attractive. (technical difficulties)

BM: Which static analysis tools do you already use?

BR: We use something called Sonar, which has multiple tools in it. It is common to run test-coverage measurement. I actually don't remember the name of the component tools. It's good for finding complexity in software, like referential loops and things of that nature. Bad style, non-conformance to coding standard, processing methods that are large or complex.

BM: Do you use it regularly?

BR: We use some of it regularly. It's not well integrated in the workflow. We do have a tool that allows us to look at the code that's written, we also have tools that makes it possible for us to do this on our work stations. Sometimes we use it, sometimes we don't. I tend to enable as much of this as I can in my IDE, so that I get warnings as I write code. And for the most part that is stuff that is very useful, because then I just fix things as I write them t fixed, when I commit the code. (Very noisy, difficult to transcribe) It's not very systematic, or I guess, if it's quite easily available.

BM: Have you used any static analysis tools to find specific security defects/weaknesses?

BR: No.

BM: Do you fear the impact of implementing additional security tools on the workflow of your team?

BR: I actually don't know how to answer that. Because if the tools are non-obtrusive in itself then I don't fear it at all. What I fear is if they make it necessary to engage mentally a lot in the tool, as to the messages it uses then I would be reluctant to use it. Getting feedback about 'This is a bad practice, this is a bad practice' and do that continually, that would be almost exclusively good.

BM: Are you prepared to act upon the findings of the security tool?

BR: That actually depends on the findings. So, for the most part the answer would be yes. And then for most of these tools there is a point above of diminishing returns. So for instance, let me take an analogy. Test-coverage - we tend to like high test coverage. But there are always cases that a test coverage is counted as number of lines covered by at least one test. Which basically means that if you test for the most common happy "day scenarios", your code will probably not break, most will be fine, which is good. But even if you test all your code, all the happy day scenarios, you'll probably not get more than like 80 or 70 percent test-coverage. There is some code that is initialization code that is not run in testing, or more commonly you have error situations and handling of error-situations that may be quite rare and writing code for every one of those is just too boring. So it doesn't happen. And that's one thing. Another thing is that you have all kinds of input parameters and permutation there of that you actually don't test for and don't measure coverage of either. Because that would be complex parameter settings and the space of complex parameter settings for the things we do is actually quite big. And even starting to dig into that somewhat is in itself very complex, so we don't do it. And if we found that it was something that was worthwhile because we did a lot of that and we've got some particularly interesting situation then we would do that. But usually we just say 'Ok, now it's enough. We have 50-70% coverage, it looks good.'. (? at 12:32) is mostly secure (? at 12:35), we won't. So it would be the same kind of logic I would suppose that we would use on a something more directed to security testing. If I am really really really obscure and the impact is not that high then it probably would be ignored. But we would have to see how the tool actually works to make... to form an opinion on that. It is hard to say something without actually having looked at the tools and I haven't looked at any of the tools yet.

BM: What challenges do you foresee?

BR: Challenges? First of all, just practical challenges. Making the things actually work, that usually is the worst thing. The hassle-factor is not to be underestimated. So if it actually works, it works in the IDE or it works in the build server, we will get regular reports, then I will come to my no. 2 fear, which is that we end up in a situation similar to what we have now with Sonar. Which is OK but we just don't use it very much. Which means that good suggestions that we should have taken into account while just actually writing the code is not taken into account. That would be my second largest fear.

BM: Does the team have the skill to use the static analysis tool?

BR: Yeah, the developers in the team have the skills.

BM: Which would you prefer, an IDE integrated static analysis tool or a report generated by a static analysis tool after build?

BR: I vote for both. If I could choose, I would choose the former (IDE), because that gives more immediate feedback. But Sonar has very nice feature and that is that we can see development of metrics for the whole team, which is much more difficult if you have it just in the IDE. So I would actually prefer to have both.

BM: Both combined or you would equally vote for both?

BR: When you sit and and work on something you work on a branch and a code repository and that branch is unique, no one else has the same as you, because you are the one developing on it. Which means that if you can get feedback on what you are doing, while you are doing it, that means that whatever hints, whatever advice the tool can give you, you can take into account while you are in "the flow". So you have developed the mental model of the system you are working on, have it all in working set memory and you get the suggestion and then you testify it. Which is very good way to work. It's much better way

to work then first get to the build server, have a unique branch at the build server, perhaps even a rather lengthy run because we have many components and it doesn't make sense to check them all at the same time. That means that the feedback is not as immediate, you are not in the flow, you need to sit down, with a list the issues and that is much more cumbersome to do.Still, it does have value. Because again, maybe your IDE doesn't support it, maybe someone else can be making stuff and the two of you who are making stuff write on top of each other and you didn't see what happened when you did the merge, but the stuff actually compiled, got merged and there was an issue. So they don't solve exactly the same kind of problem, right? That's why it would be nice to have both, because some things are quite easy to do in one way or the other, but again, if I had to choose I would choose really good IDE support.

BM: But you all use different IDEs, so it will probably be a little bit more complicated to implement it on that level?

BR: Yes, I don't see that changing either. Not anytime soon.

BM: Which agile methodology does the team use?

BR: If I were to say a word, I would say rainbow. Does that makes sense for you?

BM: I know Scrum, Kanban...

BR: We use a combination of Scrum and Kanban, I would say. And it depends what kind of mode we are in. Right now, I think we are moving towards more traditional Scrum-like development where we have a set of tasks and scope, then it's nice to make some estimates. How long it should take... Then we have the team do it in a sprint. The period prior to that, we were much more in Kanban mode, where we had a bunch of tasks and we just do them and we coordinate between us on those tasks. BWe have some fields that are more or less functionally separate, such as business support system integration and hosting and

setup and infrastructure. They are actually mostly separate and we have separate groups of people working on them. Which means that we have now small groups and then we do coordination every two weeks, make sure that we go in (? at 20:16) the same direction. It could be Kanban to, but it tends to make you daily meetings to be long, because you need to do a lot of coordination, which is not the purpose of those meetings.

BM: How often do you release versions of the product?

BR: Honest answer: I don't know. The infrastructure is mostly all done now, it's been a long time since we've released a new version.

BM: What was the last time you did a release?

BR: I would say some time last year. There has been so much infrastructure work. That is changing out, we will roll out in Sweden. So if you ask this question again in a month or two, I will probably give you an answer which is weeks or something like that. But at the moment we haven't done it for a long time.

BM: OK, so recently you've been focused on the infrastructural setup, but in the future you are expecting to release software more often than you do now.

BR: Yes, the whole purpose of doing the infrastructure the way we have done it is exactly to do that. So that we can have a development process with much more frequent releases.

BM: How do you measure developers' efficiency, if you do?

BR: We don't have metrics. So if people feel productive than they are probablyare . We don't actually measure it.

BM: Because sometimes these agile methodologies, these approaches, define also developers' efficiency and ways how to measure it and some teams that are implementing these methodologies use also these tech-

niques, so that's why I wanted to...

BR: We don't do that. In Telenor Digital we have had very varied experience with that kind of thing. We started... it's been very difficult to measure. Because we used to do estimation on stories and then we tried to dial in the estimation and after a while that sort of worked. And then we tried to measure focus factors and things like that, to see how efficient people were with respect to the estimates they made. But it turns out to be extremely difficult. Because how focused are you, on a scale from 1 to 9? And if you start factoring in, you're focused or need to focus on a few estimates, that makes your estimates more precise. But it doesn't mean the deviations from estimates is in kind of indicator of your activity. So, the answer is 'I don't know'.

BM: My next question would have been how do you estimate the team's velocity per iteration, but you probably don't do that as well.

BR: No, we don't. Basically the same as answer for estimates estimate. I just answered. So I think the best measure for that would actually be something like releases per week or month or something like that.

BM: Not only as an estimate of a developer, but as an estimate of the whole team's work?

BR: Measuring individual developers to my mind does not make sense. That's my opinion. I don't know how to measure it.

BM: Could you provide data or some project burndown charts? But you probably don't have that either...

BR: No, we don't.

BM: Because at the end of the project I would like to compare somehow how the implementation of the tool had an effect on the team's work, how it affected their efficiency.

BR: Yeah, I think you would need to ask basically. We don't have any

kind of objective measurements on that and again I don't know how to make those measurements.

BM: But you do have iterations, right? Sprints maybe, in terms of Scrum?

BR: Yeah, it looks like that. So for the last two... last month we had two week sprints. Looks like we're going to do that.

BM: Two week sprints? That's the usual time length of one sprint?

BR: Yeah, that's been usual for the last four weeks. Looks like it's going to be like that for some time now.

BM: What sequential processes are contained in or describe each iteration? Do you have any specific set of sequential processes like development, testing, review, code reviews after development? Do you have any specific tasks that go...?

BR: The sprint itself is... first we do some kind of scoping, we look at relevant tasks and also see if they are feasible at this time. Do we have the necessary ingredients and if we don't then we don't do them, but if we do we do them. And then we just work on the tasks. And the tasks are complete when they are either testable, in the test environment or they are actually implemented and in production. So code is committed to the main branch of the repository after code review and basically two people need to agree that the code is fit for being merged and then it's merged. That is a continuous process, it's not something we do in a particular phase. Also we have continuous discussion on the code as such.

BM: And do you have any validation practices maybe after...?

BR: What do you mean by validation?

BM: Validation... like is a task validated after it is supposed to be finished, after its status has changed from 'in progress' to 'done'. Does it go immediately to 'done'? Does it go in 'testing' and then to 'done'?

BR: Please repeat the question.

BM: At the beginning of each iteration probably you define some tasks. You set some scope, you define tasks, problems and you start developing them. You have two weeks to complete some set of the tasks and then my questions is: 'How does the status of these tasks change throughout the iteration?'. When developers think they have completed their task, do they change the status just to 'completed'? Or do you have any 'validation' maybe in between?

BR: No, we don't have any validation. The tasks are... We do testing in many stages. First is of course unit testing, or first is that we just talk through the design. Then we do unit testing. We try to keep coverage pretty good as I said. Which means that when we put stuff into production environment we have acceptance tests that are usually a few happy day scenarios, e.g. testing provisioning of a simulated user. And if that goes through then we say: 'OK, now it's done'. We don't try to fuzz it (fuzz testing), we don't aggressively look for mistakes. We are basically optimistic. Which is not necessarily a good thing but that is what we are.

BM: Yes, I understand. But you do some testing as well.

BR: Yes, we do. We have multiple phases of testing. We have a rule that is not strictly enforced, but we do have a rule that things that go into production should be acceptance tested. Which means that we run a test on a simulated production environment with the happy day scenarios and it should actually automatically verify that the behavior is correct.

BM: Yes. Does the team have any specific documented coding practices?

BR: Yes, we do. In practice, we only have for Java. That is the main language we use. Not the only one, but it is the main language. And then we basically use the common standard practices for Java and then there are some additional documents for Telenor Digital that

we look to. We don't enforce those very strictly because they are actually very strict and very detailed. But we do look to them and we... I would say we use most of them. And they are also easily checked automatically, which is one of the reasons that we tend to follow them.

BM: They are checked automatically?

BR: Yes, many IDEs check conformance to this standards automatically.

BM: Do you practice pair programming? You do code reviews as I understood.

BR: Only very occasionally, usually we don't do pair programming. It does happen, but it's not common.

BM: But you do code reviews?

BR: Yes, we do.

BM: Do you practice refactoring of the code?

BR: All the time.

BM: Which OS platform do you use?

BR: OS X, Windows, Linux.

BM: Which IDE do you use?

BR: IntelliJ, NetBeans, Emacs. I think there is an eclipse user, I am not really sure. There is something called Sublime.

BM: Which programming language do you use? Mostly Java, but...

BR: Mostly Java than JavaScript, fair amount of shell script,some Python. I think that covers most of it.

BM: Which continuous integration tool do you use?

BR: Jenkins.

BM: Do you have a separate testing, staging and production environment?

BR: We will really soon. That's part of the goal. That's very important goal for the infrastructure project.

BM: But you don't have separate environments for now?

BR: We might actually have that. Either we have it or... Let me take that back. We do have it, but I don't know if anyone are actually using them. But that should happen this week or next week or something.

BM: How do you measure the quality of the software?

BR: Basically we don't. We look at what Sonar tells us, it gives some kind of metrics, some conformance standards, some not many stupid things. But that's the only thing we do, that's only advisory and we're not following it religiously.

BM: So you don't have any quality assurance practices I guess.

BR: Nothing that I haven't described.

BM: Which testing tools do you already use, apart from the static analysis tools?

BR: We base everything on JUnit, which is originally made as a unit test framework. But it's just something that runs Java code, so it has been adopted for doing other things too. So we use it both for unit testing, but also for acceptance testing. So in that acceptance testing mode it runs tests towards a fully operational cluster and it runs actual interactions as seen from the clusters point of view and it reports them back as JUnit test results, so that we can look upon them using tools for for that kind of thing. And it can also run it automati-

cally, so that an automatic build breaks when an acceptance test fails.

BM: Which security testing tools do you already use, if you use any?

BR: None.

BM: At which point in the SDLC do you check for security vulnerabilities?

BR: I would say all of them except that we don't have a separate review for it. We look for it during design, try to avoid them, that is of course the most important part. Then we just try to avoid bad practices while coding, but we don't explicitly look for security flaws. We just don't do obviously stupid stuff. It's not any kind of formal process.

BM: And how do you usually test you software for security vulnerabilities? Do you do that at least once a year, maybe?

BR: Actually no. We probably should, but we haven't.

BM: I was thinking more of external testing maybe?

BR: We haven't done anything like that. There is room for improvements.

BM: Yes, definitely. If there wasn't, what would've been my job here. I wouldn't have a project.

BR: Are there any specific security vulnerabilities you would like to address?

BM: Well SS7 is of course a bit old and pretty bad, we should probably be systematic when it comes to automatically look for obvious exploits when it comes to our build interfaces and we don't. We haven't made a really good set of barriers for getting against insiders. That is improving now, with the new setup. But it's not really very good there either. Also we should do test disaster recovery, because we haven't done that. And there are some design issues that are pretty bad too. When it comes to securing secrets and sensitive logs. There we know we have vulnerabilities by design because we haven't made

much effort to avoid them. And that is something that could improve things a lot. It would be really naive to assume that we will not have breaches, so damage control and damage reduction would probably be good things to design in and at the moment we haven't.

BM: Most of the security vulnerabilities you just mentioned are not directly related to the coding practices.

BR: True.

BM: Do you think that you have in mind some kind of security vulnerabilities that might be caused by coding practices that you would like to address?

BR: What I would probably like to do is to have a set of secure attacks performed against our system on a regular basis. Preferably every day, preferably automatically. If we get the funding we will do so. Because unless we have pressure for change it's probably gonna be difficult to change efficiently and at the moment we really don't have that. We have this assumption that everything is going fine, but that's really not something that is inducive to the will to to change a lot.

BM: I have one more question. Regarding the adoption of static analysis tool, in you opinion, what are the important factors necessary for adoption of a static analysis tool for your team?

BR: Yes, I think I answered that previously. Getting it working, so we can actually get this fit for the benefit it gives us on short term. And the hustle factor. That's the most important thing. If it doesn't give some benefit on short term and it has no hustle then we just won't use it, simple as that. And then if it gives some benefit, then we can start on increasing that benefit. That would be my approach to this. Some small benefit both in time and then develop on that.

BM: That would be the key factor.

BR: Yes.

BM: Thank you very much, that was all from my part.

WHEN:	26 MAY 2016 at 09:00 AM
WHERE:	Online
ATTENDEES:	Developer 5, Bisera Milosheska

BM: What is your job title?

HR: I think it's senior developer or senior software engineer.

BM: How many years of programming experience do you have?

HR: I guess I was... 18 or 20. I've been doing development since 1998, so 18.

BM: Have you had any previous experience with security-oriented tools?

HR: Not really, we hadn't. Of course at the university where we studied security protocols, so we worked with this logic of designing security protocols. I think it was BAN (Burrows-Abadi-Needham) logic, it was called. Burrows and... there were three guys that had developed a logic for reasoning about security protocols. I think they used it to prove some errors in the Keberos authentication protocol. From that I have not really used much tools.

BM: How familiar with security vulnerabilities are you? Let's say on a scale from 1 to 5.

HR: Well I get knowledge about the major ones that are released in the news media and publications, but other vulnerabilities I probably don't know that much about.

BM: So how would you estimate your knowledge?

HR: I guess it would be in the middle of the scale, I have some knowledge but it's not a topic that I pay that much attention to. I notice the major ones, but...

BM: Maybe 3 or 4?

HR: Yeah, 3 I guess.

BM: How safety-critical is the software you are developing?

HR: Well it's not really that safety-critical. I mean we're developing phone applications or phone service but the kind of critical part that you are able to call emergency numbers is handled by the network operator, like Telenor Norway or the one that has the radio network and these things... So you can still do that over those system if our fails, so in that sense is not critical. But of course it would be very irritating for our users if it failed. And of course, people trust phone system and messaging system and so on and they expect that we keep their privacy and their information. But that part is most critical.

BM: Yes, regarding the sensitivity of data you are storing.

HR: Yeah.

BM: How do you feel about bringing software security into operations?

HR: Well of course, you want to have a system as secure as possible, but on the other side there is often a cost making it secure. And when you are on a time budget you cannot postpone the security...

BM: Would you find it distracting more than helpful?

HR: It depends. It's kind of... It has to be part of the goal of the system you are building to be secure otherwise it's distracting because if you are just looking for functionality and spend a lot of time on making your system secure or safe and doing things that you are not getting paid for or the customers are not willing to pay for. And also it's a lot easier to build system when you don't, for instance, use encryption on your communication, then it's easy to debug by just tapping into the wire and see what data flows there, but if it is encrypted everything and flows much harder to debug the application during development, so in that sense developing with security in mind is harder.

BM: Which static analysis tools do you already use?

HR: We use Sonar and different plugins for that. We get removed some of the classical bugs from the application.

BM: Mostly regarding the coding style?

HR: Yeah, coding style and how we use it. And also when we used to program in C, we used this additional tool like Lint which tries to find memory leaks and discover other things in C program.

BM: Do you use it regularly or just when needed?

HR: I try to use it on a quite regular basis, not daily but from time to time you go to my code and you got the warnings and consider if those should be removed or not really.

BM: Have you used the static analysis tools, Sonar in this case, to find any specific security issues?

HR: No, not really. More general issues.

BM: Do you fear the impact of implementing additional security tools on the workflow of your team?

HR: Some of the errors you probably remove with these tools might be potential bugs anyway, that you would have to deal with anyway, so I mean might be that you actually can at the same time be using better tools for finding problems in your system. It depends a lot on the tool and how easy it is to use and how it flows into your regular workflow.

BM: Are you prepared to act upon the findings of the security tool?

HR: Yeah, I mean it kind of depends on how critical the bugs are or the problem reported are... so hard, most of the report is actually relevant but sometimes it reports things that you know you made them

for a reason.

BM: What challenges do you foresee in implementing this kind of tool?

HR: I didn't quite understand that question.

BM: The idea of the project is to implement a static analysis tool, that will discover security vulnerabilities.

HR: I mean it probably should be discussed in in addition too of course, but developers can use it on their workstations or when they are doing their coding it probably also should be used as part of the build pipeline, so that your changes don't go into the production system. Having been run through a pipeline which also includes these security analysers.

BM: Would you prefer to have it as part of the build server or as an IDE integrated static analysis tool?

HR: I think both, you want to have it on the build server, but the reports you want to fetch like you can from Sonar, you can fetch the Sonar reports into the IDE where you're working so you can see them there and strictly jump to the code part that's mentioned. Otherwise it gets kind of a bit troublesome if you have to look at from that page or from text files and then navigate to your code so if automatic navigation to the problem's part is supported by the tool then that would be good. But of course, you need to also have it in a kind of batch mode or be able to run it on a server as part of your build process.

BM: Do you think it will be more effective if we implement it in both ways?

HR: Yes, I think so. Otherwise if you will just have it batch tool, that is kind of ... You have this big bang test of security and then you get a lot of reports, while if you have it as part of your tool you get continuously the warnings and can improve on that while you are at that part of the code.

BM: Does the team have the skill to use a static analysis tool?

HR: Yeah, I think so. If you have it there and the tool is reasonably easy to use, it will start the tool point it to the code and the tool will report in a way that people understand. I guess it also is a learning process, you have to learn what the reports really mean. But once you get used to that, you easily see that 'Oh, this is that kind of report! Ok, then I need to change this.'.

BM: Yes, it is definitely a learning process. And I agree that much attention has to be dedicated to the way how the tool is implemented. It has to be properly configured so that it doesn't report a lot of false positives.

HR: Yes, right.

BM: That was all from my part. (Skipped conversation part, irrelevant for the interview questions) Thank you very much for the interview and your time.

Appendix B

Independent evaluation phase: CWE mappings

Commercial tool

The following is just an example of the structure of a CWE mapping file. The rest of the CWE mapping files are availble in the Github repository [30].

```
<mappings scanner="Commercial tool">
  <scannerCode desc="Access Control: Amazon Web Services"
      name="Access Control: Amazon Web Services">
      <cwe id="566" />
  </scannerCode>
  <scannerCode desc="Access Control: Android Provider"
      name="Access Control: Android Provider">
      <cwe id="566" />
  </scannerCode>
  <scannerCode desc="Access Control: Anonymous LDAP Bind"
      name="Access Control: Anonymous LDAP Bind">
      <cwe id="285" />
  </scannerCode>
  <scannerCode desc="Access Control: Database" name="Access
      Control: Database">
```

```
        <cwe id="566" />
    </scannerCode>
    <scannerCode desc="Access Control: LDAP" name="Access Control:
        LDAP">
        <cwe id="639" />
    </scannerCode>
    <scannerCode desc="Access Control: Weak Security Constraint"
        name="Access Control: Weak Security Constraint">
        <cwe id="285" />
    </scannerCode>

    ..................................................................

    <scannerCode desc="" name="Unsafe JNI">
        <cwe id="111" />
    </scannerCode>
    <scannerCode desc="" name="Unsafe Reflection">
        <cwe id="470" />
    </scannerCode>
    <scannerCode desc="" name="XPath Injection">
        <cwe id="643" />
    </scannerCode>
</mappings>
```

Appendix C

Post-implementation evaluation: Questionnaire

1 Post-Implementation Questionnaire

Questions	Answers				
Future use intention	**Strongly disagree**	**Disagree**	**Neither**	**Agree**	**Strongly agree**
Given a choice, we would prefer not to use the tool in any future work.	☐	☑	☐	☐	☐
We would like to use the tool in the future.	☐	☐	☐	☑	☐
We intend to start using the tool in the future for our work.	☐	☐	☐	☑	☐
We will dedicate a part of our sprint time intended especially on solving issues reported by the tool.	☐	☐	☐	☑	☐
Perceived usefulness					
I think the tool will be useful in my job.	☐	☐	☑	☐	☐
Using the tool will improve the security of the product.	☐	☐	☑	☐	☐
Using the tool will substantially reduce the number of serious security defects.	☐	☐	☑	☐	☐
The advantages of using the tool outweigh the disadvantages.	☐	☐	☐	☑	☐
The tool is very effective in finding real vulnerabilities.	☐	☑	☐	☐	☐
The tool completely satisfied my expectations.	☐	☑	☐	☐	☐

Based on the tool's reports, I think that some of the existing vulnerabilities in the code which we are aware of, could have been detected if SAT was used during development.	☐	☐	☑	☐	☐
The tool will raise developers' security awareness and teach them good coding security practices.	☐	☐	☐	☐	☑
The tool will distract developers in their work.	☐	☑	☐	☐	☐
The tool will have negative effect on the team's atmosphere.	☐	☑	☐	☐	☐
The tool will have negative effect on the team's confidence.	☐	☑	☐	☐	☐
The tool will have negative effect on the team's ability to deliver.	☐	☑	☐	☐	☐
The tool will have negative effect on the team's focus.	☐	☑	☐	☐	☐
Perceived compatibility					
The tool is compatible with the way I do security work.	☐	☐	☑	☐	☐
Using the tool is compatible with all aspects of my work.	☐	☑	☐	☐	☐
Using the tool fits well with the way I work.	☐	☑	☐	☐	☐
The tool is compatible with the way we organize our work.	☑	☐	☐	☐	☐
I think it is very convenient that the tool is implemented both locally and on the build server.	☐	☐	☐	☐	☑
Subjective norm					

People who influence my behavior think I should use the tool.	☐	☑	☐	☐	☐
People who are important to me think I should use the tool.	☐	☑	☐	☐	☐
Co-workers think I should use the tool.	☐	☑	☐	☐	☐