Helena Saarinen

Security Activities Integrated into DevOps Software Development and Operation Processes

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

20.10.2022

Abstract

Author(s):	Helena Saarinen
Title:	Security Activities Integrated into DevOps Software Development and Operation Processes
Number of Pages:	70 pages + 2 appendices
Date:	20 Oct 2022
Degree:	Master of Engineering
Degree Program:	Information Technology
Instructor(s):	Ville Jääskeläinen, Principal Lecturer

Web application security and safety are important issues for all digital service users. The speed of application development cycles, constant changes, and continuous integrations cause many challenges to ensure and maintain web application security. Web service users have often unnecessarily worried about security, because in many cases automated scanning tools can find most of the security vulnerabilities [1]. Web applications must be built and designed to prevent intentional attacks and protect users from exposing confidential information even if they use insecure actions [1].

The Agile application development model is widely used for web service development. In this model, changes are often made to the service, and ensuring application security at the end of the process slows down the release cycles and increases costs. In addition, correcting security findings at the end of the development process is always difficult and more expensive.

The goal of this thesis is to learn more about web application protection methods by reviewing system security guidelines, best practices, and how these can be implemented in the agile application development model process at different stages.

This thesis is based on the literature which incorporates application development methods, security practices and security verification processes. Following the conceptual knowledge base, the case company application development and operation processes' current state evaluation was executed using the BSIMM framework. It was found that in case company Security Testing practice area activities' maturity level was remarkably lower than companies used in the comparison. The final proposal includes 20 high and medium level security activities to improve the case company application development and DevOps processes.

Keywords: web, application, security, agile, DevOps, DevSecOps

Contents

List of Abbreviations

API	Application Programming Interface
BSIMM	Building Security In Maturity Model
CI	Continuous Integration
CM	Configuration Management
CSRF	Cross-Site Request Forgery
DAST	Dynamic Application Security Testing
DOS	Denial of Service
DSM	Design Science Method
DSR	Design Science Research
DSRM	Design Science Research Method
IID	Iterative and Incremental Development
NFR	Non-Functional Requirement
SBOM	Software Bill of Materials
RUP	Rational Unified Process
SAST	Static Application Security Testing
SDLC	Software Development Life Cycle
SSDLC	Secure Software Development Lifecycle
SQL	Structured Query Language
XSS	Cross-Site Scripting

1 Introduction

Web applications today are mainly implemented by using the incremental Agile development model, where separate service functionalities or features are built in short development cycles. In this model, the software is gradually developed by adding new functionalities, which allows one to gather feedback from customers and make changes more flexibly than other software development models such as the waterfall software development model. [2]

Web applications security and safety are important issues for all digital service users. The speed of application development cycles, constant changes, and continuous integrations cause many challenges to ensure and maintain web applications security. Web service users have unnecessarily worried about security, because in many cases automated scanning tools can find most of the security vulnerabilities [1]. Web applications must be built and designed to prevent intentional attacks and protect users from exposing confidential information even if they use insecure actions [1].

Application version testing and releasing are performed at the end of each development cycle, and therefore one needs to use systematic version control, test automation and adequate regression testing [3]. Very often there is not enough time for a detailed risk assessment or application security tests. [2] Rapid development cycles combined with inadequate security testing will, in the worst case, compromise the system security.

1.1 Business Challenge and Objectives

Agile application development model is widely used for web service development. In this model, changes are often made to the service, and ensuring application security at the end of the process slows down the release cycles and increases development costs. In addition, correcting security findings at the end of the development process is always difficult and more expensive. The case company has also noticed that verifying application

security by using Agile development and DevOps processes is not always reliable.

1.2 Thesis Structure

This thesis is organized in 8 sections. The first section explains business challenge and the objective of this research. The second section briefly presents utilized research methodology and its usage in this thesis.

Theoretical background is divided to four main sections 3, 4, 5 and 6. Section 3 describes web application development and operation related concepts, such as software development models, DevSecOps operations, security incidents detection, monitoring, maintenance, and operations. Section 4 presents web application security risk management key elements. Web application testing and validation methods are shown and discussed in section 5. Section 6 presents 10 design guidelines, security practices in three categories and best practices in software security.

Section 7 presents the case company's current application development and operation processes and suggested improvements to the process. Theoretical knowledge, DevOps process maturity level and BSIMM software security maturity level evaluations are utilized in creating suggestions for process improvements. Final section 8 summarizes the thesis conclusions and presents future research proposals.

2 Research Method

In this thesis the research method Design Science Research (DSR) is used as the research method, because it is suitable for information system (IS) processes related studies and it provides a structured and systematic framework for conducting, evaluating, and publishing research. [4] DSR methodology objective-oriented approach was selected for use in the thesis.

Design Science Research Methodology (DSRM) framework has been created to solve problems in processes and organization. Information Systems (IS) Design Science Research (DSR) aims to form and develop artifacts that solve problems in an organization or a process. Artifacts create useful models, processes, or methods. [4]

DSR model process activities are build to work in sequential order but progressing in the process does not require sequential progression from phase 1 to phase 6. It is possible to start at almost any of the four phases, which are called problem, objective, design, and development and observation oriented. [4]

A *problem-oriented* approach is suitable as a starting point for research if the research idea is based on previous research or problem observations. An *objective-oriented* approach is suitable as a starting point if, for example, the system development has not taken place as planned and there is a need to find a solution to the identified problem. A *design and development centered* approach begins with activity 3, where the artifact has come from another research area and has been used to solve another problem, and now it is needed to examine its applicability in solving the present problem. The fourth approach is to *observe* the solution in use, which may eventually result in a DS solution if the researchers apply the process retrospectively and work backwards. [4] Design Science Research process model with possible research entry points is shown in Figure 1.

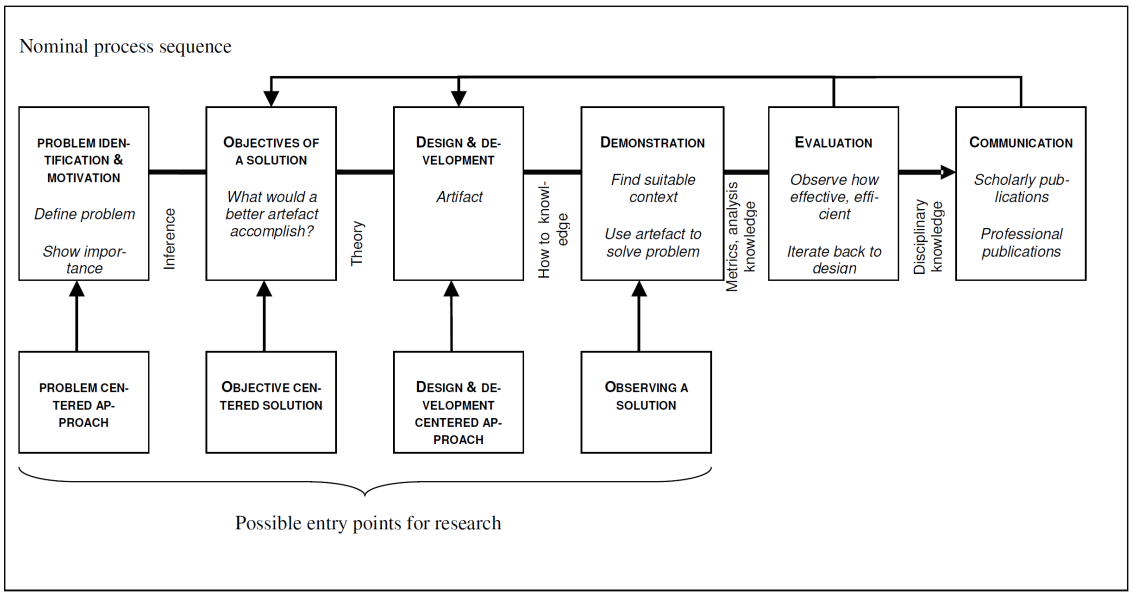


Figure 1. Design Science Research model process and possible research entry points. [4]

DSR process model contains following 6 activities 1. Problem identification, 2. Solution objectives, 3. Design and development, 4. Demonstration, 5. Evaluation and 6. Communication. Table 1 contains more detailed descriptions of the activities.

Table 1. Design science research process model activities description. [4]

Activity name	**Description**
Problem identification and motivation	Research problem definition and justification for the designed solution. Problem can be divided into small conceptual parts so that complexity of the problem is taken into account in a new solution. Solution value justification motivates researchers and client to find the best solution and to accept the result of the studies. [4]
Objectives of a solution	Problem definition leads to solution objectives definition. Problem definition can be done when the state of current problems and efficacy of the current solution is known. Solution objectives can be quantitative, where another solution is better than the

	current solution, or qualitative, where the solution solves unknown problems. [4]
Design and development	Theoretical knowledge leads new design solutions aimed at solving the original problem. [4] Developed solutions can be models, methods, technological rules or implementation principles. [5]
Demonstration	Proposed design will be demonstrated for target groups in simulation, a case study or in other appropriate activity. In demonstration it is shown how proposed design can be implemented. [5]
Evaluation	In evaluation activity observations are made on how well created solutions solve the original problem. Solution objectives will be compared with demonstrated new solutions. It is important to test and evaluate the functionality of the new solution. After evaluation researchers decide if more iterations should be made and return solution back to stage 3 for improvements. Further improvement to the solutions can also be made in the next project. [4]
Communication	Publish the results of the research, for example, as a scientific article or other publication. [4] Research findings should be shared to relevant audience via scientific articles and publications. Communication with professionals is vital to integrate findings to the knowledge ecosystem. [5]

2.1 Case Company

This study follows DSR research method stages: Problem identification and motivation, Objective of a solution, Design and development, Demonstration, Evaluation, Communication. The following list shows the stages and an activity short description. Figure 2 present DSM research study process stages and research activities.

1. Problem identification and motivation

The company has accelerated the web application release cycle, but the security verification has not been automated in every aspect, and manual security-improving practices have not been fully implemented. Static code analysis tools and code reviews have been taken into use, but they are insufficient to ensure web application security.

Companies have usually taken into use a software development model, which is customized for their organization needs, and for this reason software development and operation model security improvements must be adapted to the company's customized working model. All security practices are not necessary or suitable for all companies.

2. Objective of a solution

The research objective is to find the best security practices to improve case study company application development and operation process. Main goal is that managers can be confident that the web application meets security requirements, even though development and release cycle are accelerated.

3. Design and development

The process design and development work is based on the literature on Agile, DevOps, DevSecOps application development methods, and on the case company current state evaluation is made using the BSIMM framework. In addition, the proposal to improve the current development and operation process activities was made by using practices listed in the literature and proven to be useful.

4. **Demonstration**

   The evaluation results and improvement suggestions for the development and operation process are presented to the company's ICT department management team.

5. **Evaluation**

   Case company DevOps maturity level is evaluated by using Eficode's framework. Company software security maturity level is evaluated with BSIMM framework. Evaluation points out areas where in process is need for improvements.

6. **Communication**

   This research results and proposals are published as a Master's thesis publication in the Metropolia University of Applied Sciences.

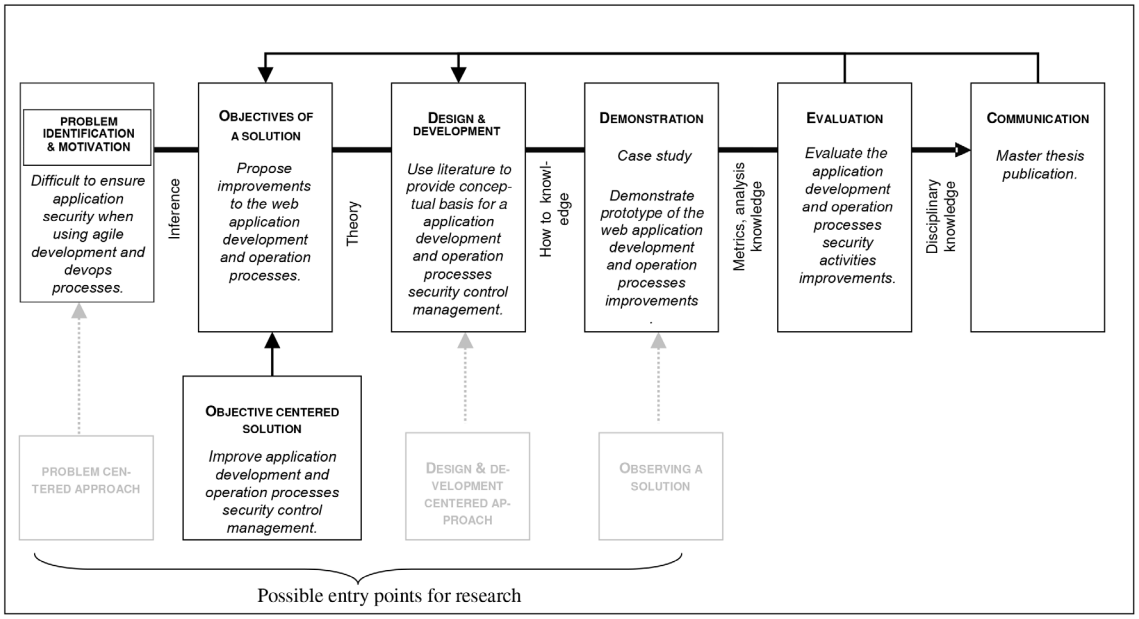


Figure 2. DS research process for the case company study. [4]

3 Web Application Development and Operation

This section briefly describes application development history, agile development methods' nature and the use of security practices in application development and maintenance. In addition, section includes information about common security challenges and the most used security practices.

3.1 Software Development

Ian Sommerville [2] in the book “*Software Engineering*” states that in systems increased complexity and misused software engineering development methods lead to failed software. More complex systems are being developed faster, and the systems are also required to have capabilities that were previously impossible to implement. Developing software by using a suitable software engineering method reduces development costs and enables the development of reliable software solutions. [2]

Royce Winston introduced in the article “*Managing the Development of Large Software Systems*” the first version of the waterfall model in 1970. He recommended implementing the steps of the waterfall model twice and delivering the software implemented to the customer only after the second round is completed. In December 1994, the new Mil-Std-498 standard was published, allowing the use of incremental software development models in addition to the waterfall model. [6] In the mid-1990s, the Rational Unified Process (RUP) model was born [6], which is an iterative software development model. [7] In the beginning of the year 2001 development process experts organized a group gathering for discussing common ground of simple IID methods and principles. Agile Alliance foundation and Manifesto for Agile software development was created. Alistair Cockburn, one of the participants, published the first book on the "*agile methods*" next year. [6]

In the waterfall model, the software development process proceeds according to a precisely defined process in stages. The waterfall model stages are

requirements definition, software design, software development and testing, integration and testing, and production use and maintenance. Each phase must be completed before the next phase can be started. Using the waterfall model is essential with embedded critical, and extensive systems. Embedded systems usually involve other hardware and mechanical parts, so decisions related to the selection of different hardware must be made before starting the system implementation, and for this reason it makes sense to use the waterfall development model. [2]

The critical system requirements and design documents should be finalized before safety and security analysis can be carried out. Using the waterfall model to implement large systems in a multi-vendor environment is recommended. [2] The waterfall model is especially suitable for the development if software requirements have been completely understood, or the project is clearly defined. In addition, the waterfall model should be used for the very large, complex, and safety-critical systems development. [7]

The incremental software development model has created the basis for the Agile software development method. In the incremental model the software is developed step by step so that the outputs of the next step are built on top of the outputs of the previous step. The incremental software development phases are requirements definition, software development and testing. [2]

In Agile software development the product is being developed gradually by adding new functionalities on every development around. [2] This approach usually improves software quality, requirements management, customers, and team satisfaction, because it enables users to give feedback earlier and testing can be carried out on an early stage. [2, 8] The weak point of the agile method is that software development preliminary cost estimation is very difficult. [8]

It is challenging to estimate or give on exact price for the software development, especially in situations where a lot of changes are made to the application requirements. [2, 9] 60% of the software production costs come from the

development costs and about 40% are the software testing costs. The long service life of the software increases the maintenance costs, and in some cases, costs can be higher than the actual software development costs. [2, 7] The new components development often leads to increasing number of errors. After development, errors must be fixed, and the application must be tested carefully. [7]

The waterfall is an old mainstream application development method, where all components are in advance designed, and implementation is monitored based on plans and agreements. It is very difficult to include everything related to application development in contracts. In the worst case, this may lead to a poor user experience, and the necessary repairs will inflate the budget and delay the application's development schedule. Agile methods provide a solution to the problem of changing requirements, emphasizing the phased development process and communication. Conflicts encountered with the technical maintenance team are resolved in the third generation DevOps development method. [10] In the DevOps development model the weakness and strength is the fast-paced release cycle. The release cycle speed enables corrections to be implemented quickly, but on the other hand, ensuring application security cannot be done quickly enough with manual testing methods. When adopting the DevOps development model, the DevSecOps method practices and tools must also be implemented.

3.2 Agile Development Model

The agile development model was developed to solve the problems of the waterfall model. Agile development solutions to the problems of the waterfall model were e.g. developing the software in small units and that, made it possible to implement changes and test the software already in the implementation phase. In addition, the content and outputs of the development iterations can be seen and monitored at regular intervals. [10] Agile methods made application development and following the progress much easier and more transparent for customers.

In the agile software development process software design and implementation are developed concurrently, and the code will be refactored as the design progresses. Continuous changes to the application, in the worst case, results in structurally poor-quality program code, which is not easy to maintain and can also lead to a deterioration in security. Reference [2] recommends regular basis improvements and structural refactoring to the program code when using agile methods for software development. [2]

Continuous integration (CI) is an important part of the agile software development practices where every developer integrates regularly developed parts of software. Automated test practices allow uninterrupted integration flow and effectively identifies potential integration issues and detects errors as soon as possible. Developers will receive a notification and a test results report after automated tests fail. Continuous integration process implementation can be carried out if one has a version management tool, automated builds, unit tests and if each developer commits their outputs to the version management system daily. It is important to constantly ensure that the development, test, and production environments are unified and in harmony with each other. [1]

Researchers have discussed how security issues should be handled on the agile development process. Some researchers think that security and agile development will lead to unresolved conflicts, and some researchers have suggested to use security-focused stories as a solution. Some companies do not use agile software development methods because of the conflicts with security and risk analysis policies. [2]

3.3 DevOps

DevOps has been developed to solve problems found in agile development, such as delays in new feature rollouts, deficiencies in application quality assurance, application components integration is not possible or new features break existing functionalities. Deficiencies in the Agile development model often

cause overruns of the budget and schedule set for development work. In addition, development and maintenance teams do not cooperate smoothly. [10]

The DevOps operating model can be seen as an Agile development model extension, as a development and maintenance teams dialogue and cooperation enabler, or as an independent development model. In the DevOps development model, application lifecycle loop flows through from the designing, coding, building, testing, releasing, monitoring phase and returns to the design phase. The development model loop-like structure reflects the necessity of cooperation between development and maintenance after application deployment because the information obtained from monitoring and customer feedback is necessary for planning the further product development. [11] Figure 3 shows DevOps lifecycle loop flow from designing phase to monitoring phase.

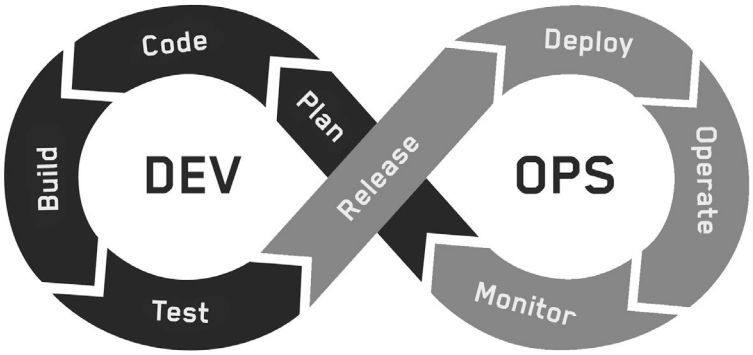


Figure 3. The DevOps development lifecycle. [11]

Lwakatare et. al. [12] the article "*DevOps in practice: A multiple case study of five companies*" described shortly that DevOps is a mind-set change to encourage collaboration between development and IT operation teams and to take all possible automated practices to use. Development and operation teams should be combined to accelerate changes delivery and to operate more resilient systems. [12]

According to Lwakatare [12] in the DevOps four key principles are collaboration, automation, measurement, and monitoring. In the DevOps operating model, the focus is especially on smoothing the cooperation between the development

team and the maintenance team. Increasing automation level is a prerequisite for DevOps development model implementation. Better resources allocation is possible with comprehensive monitoring, which detects, reports and corrects detected problems. [12] The main goal of DevOps is to encourage communication between teams and integrate development and operation specialist together to gain more rapid releases and feedback from service users. [12]

The Agile development model does not consider, for example, tools and IT environments used in application production. Implementing the DevOps development model requires changing old IT environments to virtual or cloud environments and automating all processes that require manual work. [10] Firstly, application testing and releasing must be automated to speed up the release cycle, development and getting feedback. Process automation changes the organization culture by transferring more responsibility to developers. [11,12]

DevOps development model implementation maturity in the organization can be evaluated by mirroring implemented activities to the DevOps four principles (culture, automation, measurement, sharing) realization. [11] Eficode [10] created an organization DevOps maturity model, where are seven evaluating areas: leadership, organization and culture, environments and releasing, continuous integration, quality assurance, visibility and reporting, technology, and architecture. [10]

Application delivery speed and quality are negatively affected by the problems of non-functioning cooperation between the development and maintenance team, and if the maintenance team does not participate enough in the development process. The DevOps development model focuses especially on speeding up the delivery of application versions and automating processes. [10, 12] Maintenance teams value the stability of systems and development teams value the agility of the changes flow. [12] Different goals between the maintenance and development teams may cause conflicts between the teams,

and for this reason it is important to unify the goals of the teams and remove obstacles to open dialogue. [10, 12]

The application quality improves when the versions development and delivery speed can be improved. The application quality improvements can be seen in customer feedback and a reduction in the number of errors. [12] Previously detected faults or errors in the application were not the developer's responsibility area. In the culture of the DevOps development model the developer's responsibility is to help administrators in troubleshooting and solving various problems. Developers and administrators are jointly responsible for application quality and error correction. [11, 12]

Monitoring and measuring is an essential part of the DevOps model practices. They are partly created as a byproduct of automation. Monitoring and measuring enables quick reaction to problems, learning and for example to make improvements to the development and maintenance team working methods. It is important to create the metrics so that they measure the right things. The application production efficiency must be monitored, and the monitoring metrics must cover not only the phases involved in maintenance but also the phases related to application development. [11]

DevOps development model implementation might cause challenges to achieve common understanding of the concept, difficulties related to convincing senior management, unclear responsibilities between system development and maintenance teams. Automated releasing process implementation can typically cause changes in the way the organization's infrastructure is managed. The new operation methods, technologies and tools adoption can cause challenges for the application developers and administrators. [12] Table 2 summarizes the DevOps development model benefits and challenges.

Table 2. The DevOps development model benefits and challenges. [12]

Class	DevOps benefits	DevOps challenges
Automation	The delivery speed of software changes improves. [12]	Difficulties in automating and monitoring the infrastructure. [12]
Personnel	Improves maintenance and development work productivity [12]	Unclear responsibilities between teams. [12] Adopting new tools and working methods. [12]
Quality	Improves application quality [12]	Finding the balance between speed and quality. [12]
Culture	Improves organization culture [12]	Lack of common understanding and difficulties in convincing senior management. [12]

3.4 DevSecOps

The DevOps development model has been criticized for not taking enough into consideration when it comes to the application security issues. Continuous changes and fast-paced releasing cycles can affect to the application security negatively. It is difficult to measure the application's security level. Seamless and continuous cooperation between security experts, developers and IT personnel is essential for ensuring application security. [11]

The DevSecOps development model is an extension of the DevOps development model, which takes application security into account and strengthens the cooperation with security, development and maintenance teams. The DevSecOps development model aims to ensure the application’s security before it is delivered to users. In the DevSecOps model, security management is integrated as an integral part of the development process, where the representatives from the three teams work seamlessly together in all the development process phases. [13]

The research group found 18 challenges related to the DevSecOps development model from the selected literature, which belonged to the development model 10 key categories. The identified challenges were validated based on the results of a questionnaire sent to experts. The research results showed that there were four factors that were the most challenging for the DevSecOps process: lack of coding standards (1), lack of automated testing tools (4), lack of knowledge about static testing (2), lack of communication to the DevOps team about the security standards (3). [13] Table 3 shows list of 18 challenges with their classifications.

Table 3. Security challenging factors. [13]

No	**Challenges**	**Categories**
1	Lack of automated security testing tools [13]	Secure Testing
2	Ignorance in static security testing [13]	Secure Testing
3	Communicate security standard to the DevOps team [13]	Standards
4	Lack of secure coding standards [13]	Standards
5	Inconsistence security policies design and performance measures [13]	Compliance and policy
6	Developers' resistance to integrating security protocols [13]	Compliance and policy
7	Neglecting change control in security [13]	Compliance and policy
8	Using unstable performance metrics for security evaluation [13]	Strategy and metrices
9	Translate compliance constraints to requirements [13]	Requirements
10	Lack of software security awareness [13]	Training
11	Integrate and deliver security features [13]	Security feature and design
12	Threat modelling scalability issue [13]	Security feature and design
13	Perform security feature review [13]	Architecture analysis

14	Periodic penetration test for application coverage [13]	Configuration management
15	Define secure deployment parameters and configurations [13]	Configuration management
16	Identify software defects found in operations monitoring and feed them back to the development [13]	Configuration management
17	Use application behavior monitoring and diagnostics [13]	Software Environment
18	Use of immature automated deployment tools [13]	Software Environment

Secure testing is vital in detecting security-related errors. Most of the security problems in DevOps software development are caused because automatic testing tools are not in use and static security testing is ignored. [13]

The organization creates security standards for the used technology, and these standards must be followed in the application development and the service management operations. The DevOps team is required to follow company standards for security. Problems incur if secure coding standards have not been defined or the development team has not been told clearly enough about the standard requirements. [13]

Compliance and policies play an important role in managing application risks. Appropriate security practices must be followed, and changes made to the application must be audited against them. The application development schedule pressures can lead developers into situations where they do not have the time to integrate security-enhancing components to the application. [13]

Application security objectives should be identified, and assessment should be in the organization's strategy. Incorrectly chosen or unstable performance metrics can cause problems for the organization. In addition, the organization must set for the application security requirements, such as authentication

methods, input validation, security guidelines, etc. The DevOps team converts security requirements into application requirements for each functionality or procedure. [13]

DevOps team security awareness can be increased by providing relevant security-related training. This enables better prerequisites for maintaining and complying with the organization's security standards. The training events can be held as a workshop or group training. [13]

Architecture analysis is used for security features performance evaluation. Evaluation process observe all security features and tries to find design problems. [13]

Configuration management includes application updates, version control, event handling and error tracking. Penetration testing is used to find application vulnerabilities. Findings in security testing are delivered to the development team for analysis and correction. The parameters and configurations used in version releasing must be sufficiently protected. In addition, application errors detected by monitoring must be identified and returned to the development team for correction. [13]

The application actions must be monitored to detect possible attacks as early as possible. The software installation guide must describe the software environment in a human-readable format and contain information about the necessary licenses. [13]

Using third-party libraries in application development may weaken the system's security, because often these libraries are not thoroughly tested, and therefore potential security gaps remain undetected. [14] Adding automation to security testing is part of the solution. Table 4 shows a list of 5 automatable security practices.

Table 4. Automated security practices. [14]

Activity name	**Description**
Automation of Monitoring	System-related information will be gathered, reported, and stored automatically. Information is analyzed with automatic tools. [14]
Automation of Testing	Security testing tasks are automated. [14]
Automation of Code Review	Code review is the activity where code will be reviewed automatically, and developers get appropriate feedback for the needed of corrections or improvements. Automated code review can implement by using static analysis tools. [14]
Automation of Software defined Firewall	Organization's firewall settings are managed with automated tools. [14]
Automation of Software Licensing	Automated software licensing is the activity for ensuring automatically that users use software according to the conditions set. [14]

Design review, input validation, untrusted input isolation, performing compliance requirements, security configurations and security policies are non-automatable security practices. Security manual tests, threats modelling, requirements and risk analysis must also be implemented manually. [14] Table 5 shows a list of 9 non-automatable security practices with short activity description.

Table 5. Non-automated security practices. [14]

Activity name	**Description**
Design Review	Software design review where entire software including different modules are reviewed to potential security flaws identification. The goal is to avoid finding security flaws application development later stages. [14]

Input Validation	Input validation activity perform data validation where incoming and outcoming inconsistent data will be rejected. [14]
Isolation of Untrusted Inputs	Untrusted inputs will be identified and needed security measures are performed to prevent damage. Third party library can be untrusted component requiring isolation. [14]
Performing Compliance Requirements	Performing compliance requirements need to be continuously verified. Authorities' set regulations must be followed. [14]
Performing Security Configurations	Protecting security configurations is activity where potential configurations will be identified and protecting them with security tests. [14]
Performing Security Policies	Ensuring data accessibility only for appropriate level authorization. [14]
Security Requirements Analysis	Security requirements analysis activity identifies application capabilities that prevents software unauthorized access. [14]
Performing Manual Security Tests	Manual security tests ensures that application functionalities are implemented correctly and including security tests that simulates possible attacker actions. [14]
Risk Analysis	Risk analysis is activity where security design specifications are created. Design specifications are used later for testing. [14]
Threat Modelling	Thread modeling activity identifies, describes, and classifies all threats together with different parties. [14]

3.5 Security Activities in Agile Development

The companies that answered the survey in Rindell et. al. [15] research, had adopted the practices of agile application development and combined them with various practices that improve information security. Agile development usually uses simple security techniques focused on the early stages of Secure Software Development Lifecycle (SSDLC). [15]

Ensuring software security is especially interesting to authorities and software users. Software information security is best achieved by including information security elements at the very beginning of the software's life cycle. Uniform data security metrics and strict management practices are a prerequisite for ensuring software security. [15]

Security planning could be used effectively as part of software development. Security requirements can be achieved by bringing them into the daily development work and Scrum backlog. Compliance with coding standards and code review practices are activities in accordance with the principles of agile development. Static and dynamic code analysis should not be separate activities but should be integrated as part of the code review processes. [15]

The relationship between software security and development efficiency is a complex one. The prerequisite for the evaluation is to know the achieved security level and to know the development efficiency of a specific security engineering activity. Security-related processes should not affect the efficiency of development, but neither should security be jeopardized by using unsuitable or too light data security verification processes. [15]

Security failures and threats significantly affect the costs of software development and maintenance throughout the software's life cycle. The components of secure software design are the definition security goals, requirements, programming practices, security architecture creation and design. During software life cycle, information is collected about security assurance and the functionality of security measures. Software security engineering main goal is to address identified security risks in the development phase, so that security solutions can be implemented effectively. The security planning phase offers software developers the means to comply with the security requirements set for the software. [15]

Budgeting and implementation schedule constraints of software development organizations are closely monitored. Therefore, the processes of security

planning and efficient application development are very important for the unit responsible for development. [15]

To mitigate and manage security, well-known checklists are often used to guide software design and implementation practices. The aim is to prevent latent security problems with activities in the early phase of the software's life cycle. This effectively prevents the occurrence of security failures. During the software implementation and maintenance phase, external mechanisms are introduced to increase security. [15]

Security flaws common characteristic is their persistence, and they often become software "*features*" that require ongoing application monitoring and expensive security engineering activities throughout the application's life cycle. Fixing defects that affect safety afterwards is significantly more expensive than removing them at an early stage of the life cycle. [15]

Organizations' security practices are often presented in the form of information security maturity models. Strict maturity models guide the organization and developers to implement their own customized practices suitable for the processes. Customizing the best practices to suit the organization is in line with general practices. Security-specific maturity models with checklists are not sufficient in all contexts. [15]

The SAFECode maturity model is based on a strict security risk management process, which deals with risks identified by design and coding practices, as well as strict management of third-party software components. SAFECode contains concrete and practical instructions for implementing security processes. In addition, it contains the guiding principles for building an improved software development process by using SSDLC. [15]

The DevOps development model continuous integration and implementation also means faster recovery from security failures. The main challenge is how to

combine security planning practices, maturity models and standards compliance requirements as part of the flexible development method workflows. [15]

Success in requirements management usually leads to software project success. In the requirements definition phase, security requirements are clarified, goals and criticality are defined, and business effects and risks are analyzed. In Rindell et. al. [15] research, about 50% of the respondents considered the definition of goals and criticality to be insignificant in terms of application security. This finding conflicts with security requirements. Reviewing security requirements is used less often, although its effect is generally considered effective. Using the architecture guideline, the opposite trend was observed in the study. [15]

In the planning phase application technical implementation is clarified and adapted to fit the set requirements. Risk analysis is updated with technical specifications and mitigation plan. The application security architecture is very detailed in the design phase. Security tasks in the planning phase also include threat modeling and attack surface identification. Agile development often uses information security stories and abuse cases. It is recommended to add backlog security stories containing misuse cases already at the planning phase. [15]

If threat modeling according to the recommendations is not done, then the attack surface is not sufficiently known either. It was possible to conclude from the responses to the survey, that those respondents who did not implement threat modeling did not define security configurations either. Threat modeling was perceived to be difficult to implement or overwhelming and to have little security impact. [15]

Rindell et. al. [15] research main purpose was to get more information on used best practices and expert evaluation about the significances of the practices. The research group received 62 valid responses to the Agile security work practices questionnaire. Agile security practices were divided to development process phases: requirements, design, implementation, verification, and

release. Companies evaluate all practices following five-point scale systematically, mostly, sometimes, rarely, or never used on the projects. All security engineering activities perceived impact was also evaluated on a five-point scale. [15] The research questionaries' results are shown in Figures 2 to 6 where activities significance is shown with gray color and high or very high impacted activities are marked with green color.

The requirement phase included 9 security activities, of which 4 activities over 40 % of respondents estimated to have a high or very high impact on the application's security (Figure 4). The most significant activities are establishing security requirements, business impact analysis, translating compliance constraints to requirements and reviewing security requirements. [15]

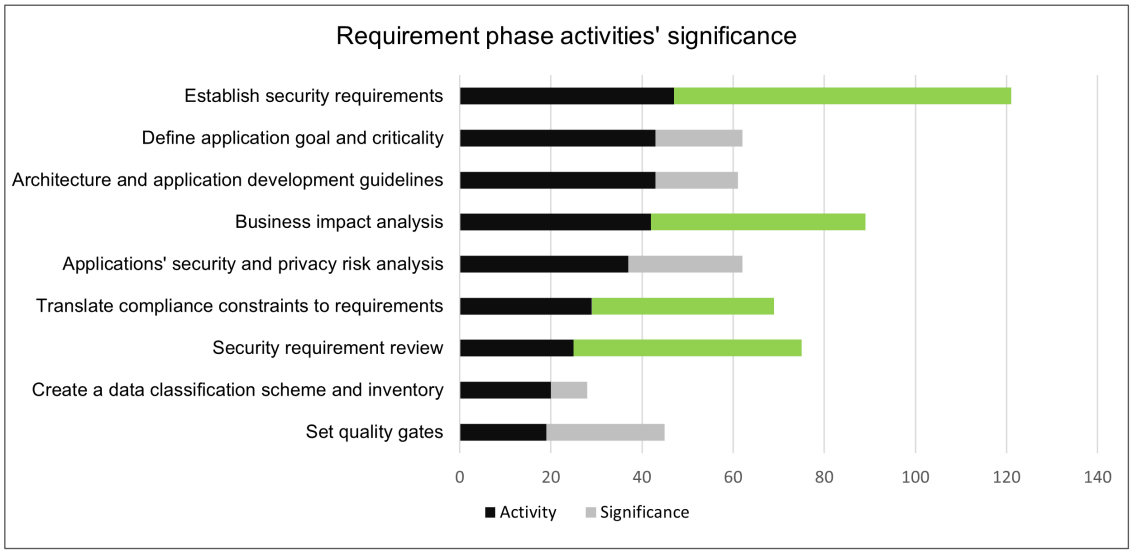


Figure 4. Requirements phase activities and their significance. [15]

The design phase included 7 security activities, of which 2 activities over 42 % of respondents estimated to have a high or very high impact on the application's security (Figure 5). The most significant activities are designing requirements and using abuse or misuse cases. [15]

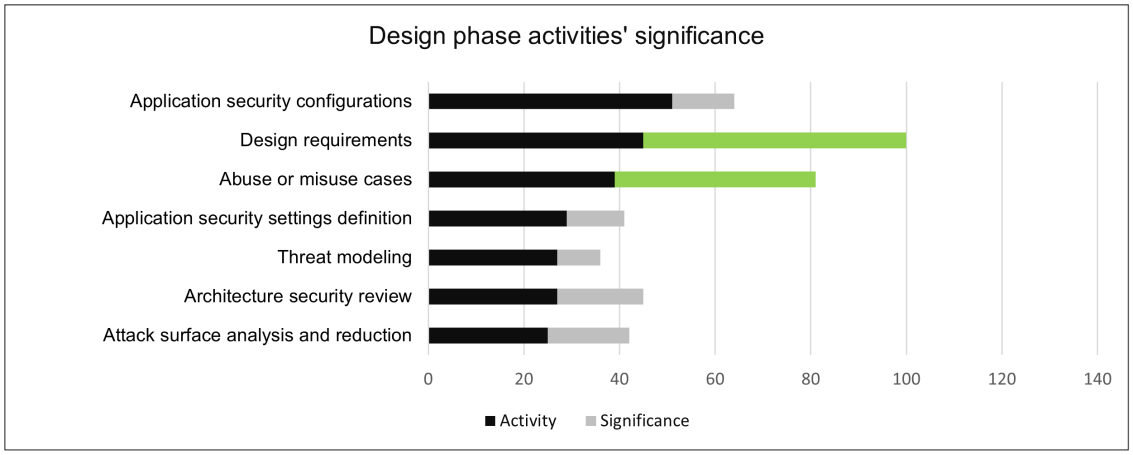


Figure 5. Design phase activities and their significance. [15]

Application development is highly automated. Various tools that ensure security should be integrated directly into development tools. The implementation phase security activities are coding standards, security documentation, configuration, and implementation reviews. According to the respondents, in the implementation phase security hardening sprints are rarely used, even though they are perceived to be important for application security. Hardening sprints are "*a security gate*" by nature which might negatively affect the agile workflow and it has affected the wider usage of this activity. [15]

The implementation phase included 8 security activities, of which 5 activities over 30 % of respondents estimated to have a high or very high impact on the application's security (Figure 6). The most significant activities are using coding standards, using approved tools, rejecting unsafe functions, reviewing security with automated tools and implementing hardening sprints. [15]

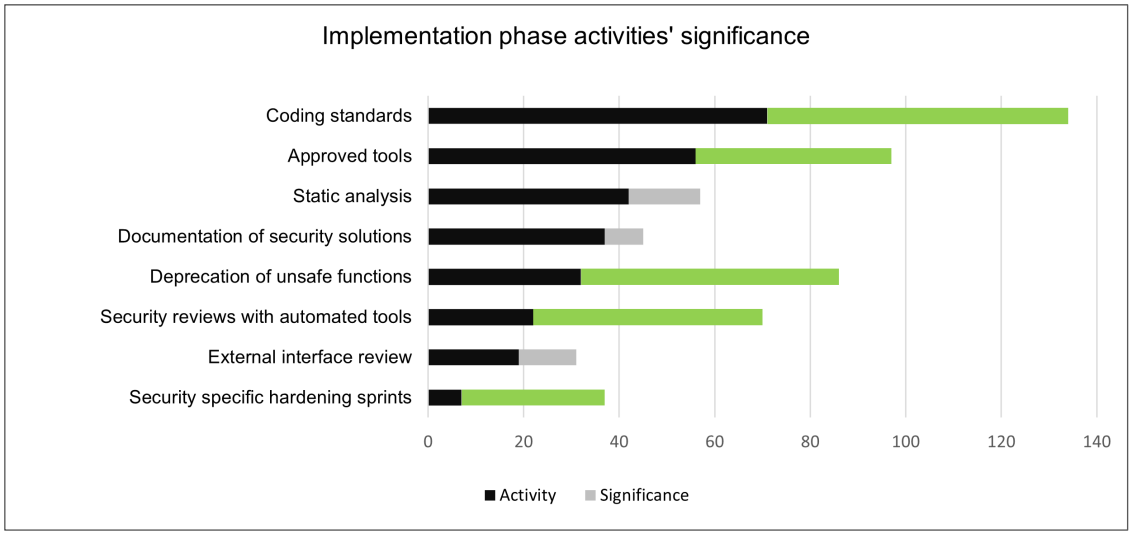


Figure 6. Implementation phase activities and their significance. [15]

In the security testing phase implementation compliance with the requirements are ensured and exploitable security vulnerabilities are prevented from entering production. Verification activities consists of security specific test cases, automated security, and penetration testing. The automation in security testing is a positive trend, but based on the research, security test plans are missing and only testing during implementation is relied upon. Traditional security testing emphasizes post-development testing. [15]

The dynamic analysis use is limited because it is considered laborious to use and it’s impact on security is considered low. The fuzzy testing use is also limited because implementation difficulties and testing period influences on agile workflow. Respondents consider fuzzy testing an effective method to improve application security. [15]

The verification phase included 9 security activities, of which 4 activities over 32 % of respondents estimated to have a high or very high impact on the application's security (Figure 7). The most significant activities are reviewing security testing plans, using automated testing tools, penetration testing and fuzzy testing. [15]

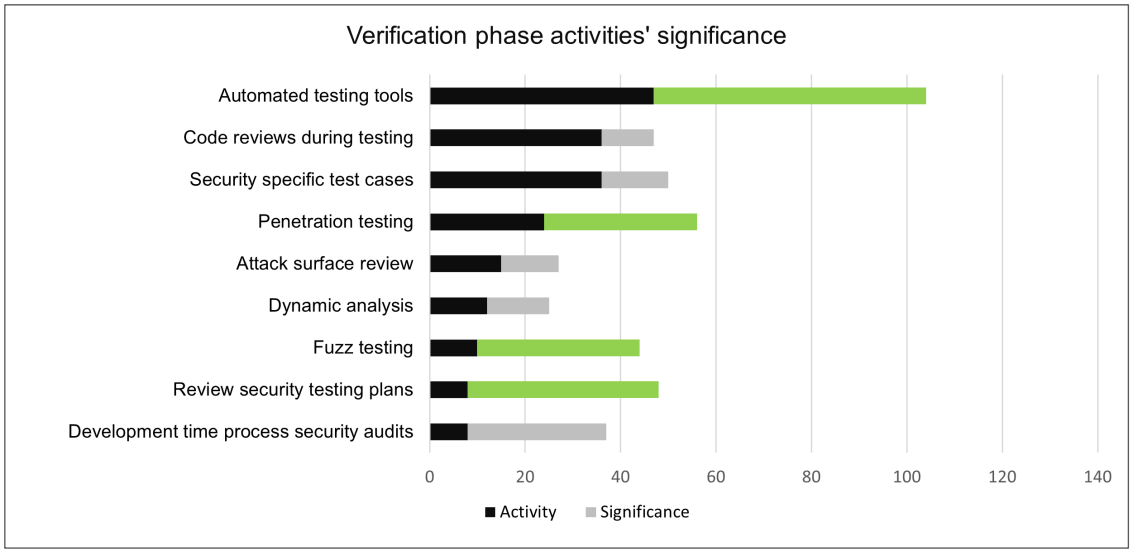


Figure 7. Verification phase activities and their significance. [15]

Continuous integration has blurred traditional life cycle model and fully automated releasing cycle, challenged us to resolve concerns related to maintaining the application. Security audits, operational mechanisms, and documentation must be in place before releasing new version of application. The respondents did not consider host and network security were a significant security-enhancing activity, even though it was the most used activity in organizations. According to the respondents, security audits such as reviews and security tests had a positive effect on the security of the application. [15]

The release phase included 7 security activities, of which 4 activities over 38 % of respondents estimated to have a high or very high impact on the application's security (Figure 8). The most significant activities are creating incident response plan, planning security patches, implementing internal and external security audits. [15]

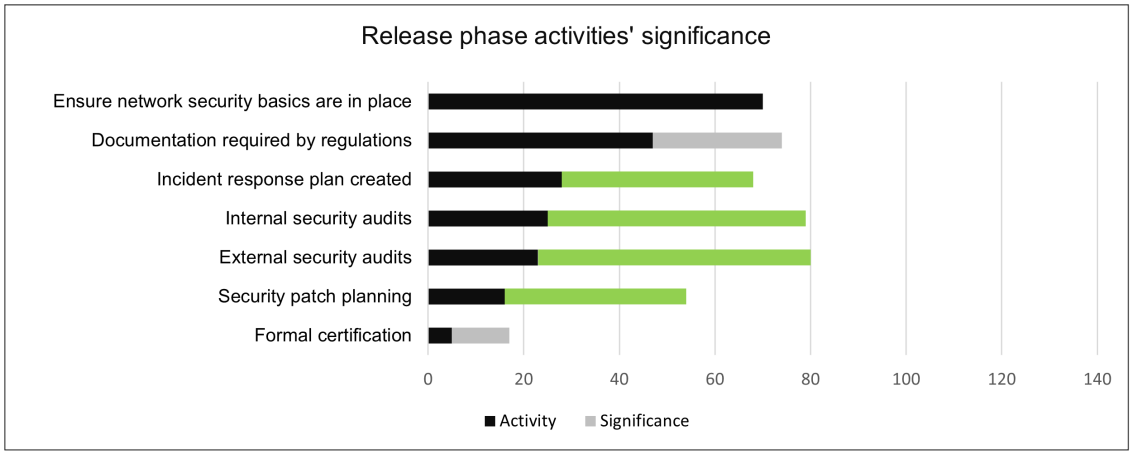


Figure 8. Release phase activities and their significance. [15]

3.6 Service Maintenance and Operations

Secure systems have three information security dimensions: confidentiality, integrity, and availability. Confidentiality ensures that only authorized users have access to the information. Integrity of the information ensures that the data on the system is reliable. Availability ensures that accessing the system and its data is normally available. [2]

Company organizational point of view security has three levels where infrastructure, application, and operational security must be considered. Infrastructure level maintains organization on all systems and networks security. Application level consists of individual application system security protection issues. Operational level enables secure operations and use of organization systems. [2]

Infrastructure level security is on the system manager's responsibility. System managers configure system infrastructure and consider possible attack vectors. User and permission management, system deployment and maintenance tasks are integral part of system security management. The system managers ensure that the authentication mechanism is working properly, and system user permissions are up to date. System software deployment and maintenance activities ensure software on all components updates and proper configuration.

A Few of the most important tasks that the system manages are system recovery, attack monitoring and detection. [2]

Software evolution process relates, and is part of the change management process, and consist of stages change identification, change impact analysis, release planning, and change implementation. Software evolution process is an on-going process throughout the software or system lifetime, and it is a part of request related change management process. [16]

In many cases maintenance costs are greater than software development costs. For instance, with the re-engineering it is possible to reduce maintaining costs and rebuild the system more preferably than to develop a new system. [16]

The maintenance activities do not affect the system or application architecture. Usually changes cause modifications for existing software components or there is a need for a new component. The software maintenance types are bug fixing, modifying software, and implementing changed requirements or new requirements. The software bugs are urgent when a fault prevents the use of a system or normal operations cannot be continued before repairs have been done. Urgent changes must also be implemented if the system environment is working unpredictably or if business changes need to be published rapidly. In previous cases' changes can be implemented and published quickly, bypassing the usual steps required by the software engineering process. [16]

Continuing changes to the software, at the worst case, results in structurally poor-quality program code, which is not easy to maintain and may adversely affect the implementation of new functionalities. Sommerville [2] recommends regular basis improvements and structural refactoring to the program code when using agile methods for software development. [2] Refactoring is an essential part of the agile software development process, it improves program quality, software structure and reduces maintenance costs, complexity, and the number of problems. Typically, agile method software improvements are

decided by the programmers. Refactoring is a continuous process that doesn't cause any overhead costs to the project. [16]

Continuous integration means automated builds and tests. Automated quality process detects errors as soon as possible. If automated tests fail, the development and operation teams will receive a notification and a test results report. [1]

The agile software development testing techniques improve the development team and customer confidence, that all planned features will be completed at the end of the development cycle. The developers get instantaneously feedback if acceptance and unit tests arise issues related with regression tests. Multiple software version builds a day and automated test practices allow uninterrupted integration flow and effectively identify potential integration issues. [1] It is impossible to do continuous integrations every day without using configuration management (CM) tools. [2]

The agile development process approach causes problems especially in cases where the development team uses the agile method, and the evolution team is using a plan-based approach. In this case, the evolution team needs to have detailed documentation from the development team and usually they do not produce all necessary documentation automatically after each increment. Another problem, in this case is, that in the agile method refactoring is an integral part of the application development process and therefore refactoring should be done when a handover is taking place. [16] Engblom [17] recommended, that the systematic implementation of software documentation is important, in order that the new features can be transferred reliably into the production environment. [17]

Eriksson’s agile transformation process testing was a problem, because the features were accumulating too large entities and led long and too heavy testing periods. One solution was that they reduced the size of feature packages to be published. [17]

Configuration management consists of four related activities; version control, system building, change and release management. The software version management keep track of system components versioning and ensured that developers tasks do not affect each other. Software building keeps track of program components, data and library versions. Change management keeps track of change requests and control change related issues. The final area is release management which keeps track of system versions. [2] Continuous integration process is a related sub-process of the configuration management under system building activity. [16]

4 Web Application Security Risk Management

Regardless of the used development method, application security controls are defined already in the design phase, and because of changing business requirements and environmental threats, it should continue throughout the application's lifecycle. Security requirements realization needs to be monitored through development process implementation and verification phases. Application deployment can affect security needs, so it is necessary to consult compliance and operation teams. [18]

Workflow includes following steps: [18]

1. Identify threats, risks, and compliance drivers.

2. Identify security requirements to combat threats and risks.

3. Communicate security requirements to the development team.

4. Validate implemented security requirements.

5. Audit used polices or regulations compliance.

Application Security Control Management inputs consist of design principles, coding practices, legal requirements, policies, standards, feedback from incidents. Outputs are artifact audits, task implementation and validation. In Figure 9 application security control management components are shown based on the vision of the SAFECode community. [18]

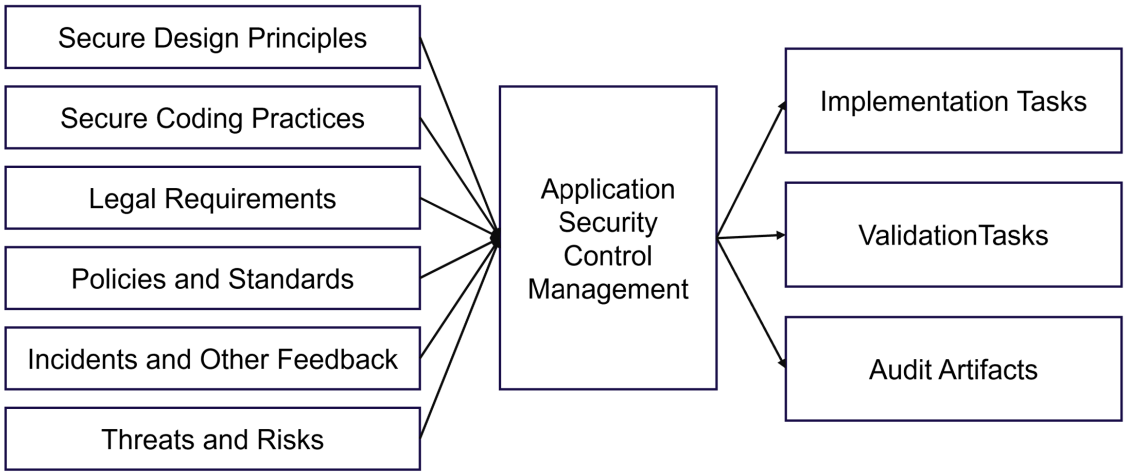


Figure 9. Application Security Control Management components based on SAFECode community. [18]

4.1 Architectural Design Risks

Reference [2] pointed out that improper architecture causes difficulties to maintain the system information confidentiality and integrity. Incorrect design solutions may also compromise system availability level. [2] All architecture decisions should be done advisedly because it is very hard to change system components later. The architect is ultimately responsible for the technical and high-level solutions, so it is necessary to compare different solutions carefully before implementation. It is also recommendable to explore previous solutions before starting to plan a new solution. [19]

System architecture security design should follow main principles; how to protect system critical components from external attacks and how to distribute components so that possible attacks have minimal impacts to the system security. If critical components are situated in one place of the system, you must build protection around the components and situate components to separated layers on the system. [2]

Layered architectural protection approach consists usually of the following protection levels; platform-level, application-level, and record-level (Figure 9). Platform-level protection layer handles system authentication, authorization,

and content integrity management. Application-level protection layer handles application accessing and transaction management, and database recovery. Record-level protection layer handles records accessing authorization, encryption and integrity management. Record encryption prevents data from being displayed in the file browser. The multilayer protection system might affect system usability and performance requirements. [2]

The agile development process concentrates primarily to functional requirements and because of this the developers sometimes ignore system quality recrements completely. The architectural approaches can be integrated to agile process if the development team is able to pay enough attention to functional and non-functional requirements. [20] According to [21] non-functional requirements are often ignored because customers typically concentrate on user-related functionalities. This might cause a lack of technical capabilities such as application performance, scalability, maintainability, portability, and security. [21] In Figure 10 a layered protection architecture example is shown.

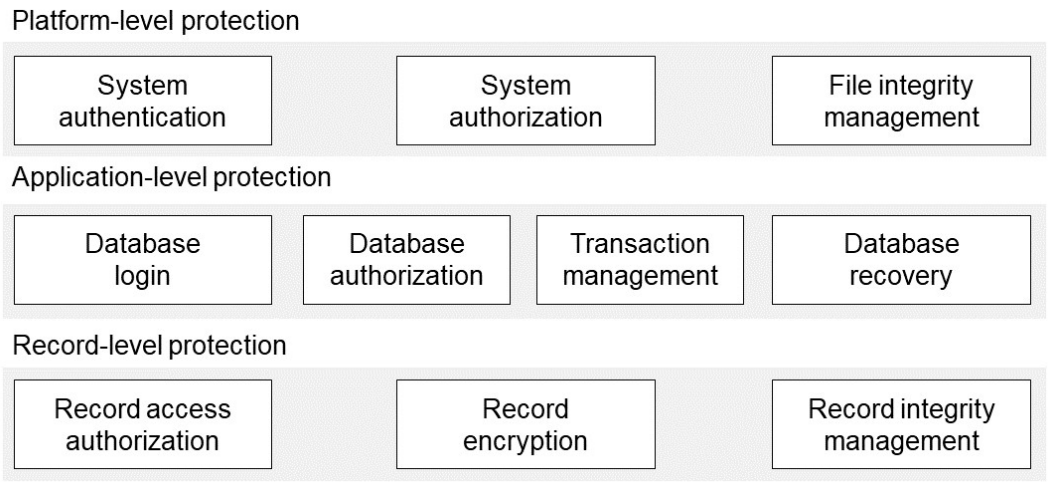


Figure 10. Example of the layered protection architecture. [2]

The software developers play an important role in combining architecture principles and agile development approaches. Communication between architects and development teams needs to be strengthened to maintain a

common understanding of system requirements and architectural solutions. If architectural decision making takes too long, it might lead the whole project to be in chaos. [20]

4.2 Security Requirements

Requirements engineering is a process that identifies stakeholders and their needs for the application. Stakeholders' different perspectives have significant impact on requirements of the software. [2, 22] Business, user and technical requirements together create the software requirements. Business requirements describe why the software will be developed. User requirements consist of functionalities descriptions and describes how software should be working. Final area of requirements are technical requirements where software quality requirements, also known as non-functional requirements, must be defined. [23]

Kauppinen [23] emphasized that the systematic approach to requirements engineering process plays a key role in the system development. The requirement engineering process begins before the project starts and continues throughout the whole project and it continues after the actual project, to see how the software works and meets the needs of users. Requirement engineering is a continuous ongoing process, and it should be a similar type of process in waterfall and agile model projects. [23]

Requirements changes must be assessed in relation to existing requirements and at the same time must take into account the constraints imposed by the system architecture. [22, 23] Requirements can be removed from the project scope in both models but in the waterfall model, it is not possible to make changes easily during the development process. [7, 22]

Iterative requirements engineering has at least two key benefits. The main benefit being preferable relationship with customers, which can be seen in software quality and stability. Secondly customers involvement to the software development process improves requirements comprehensibility and leads to a

better understanding of product features. Especially in situations where many changes must be done for known technical reasons, or if an organization wants to explore new technologies and take them into use. [21]

Agile requirements engineering approach causes several challenges such as increasing costs, schedule changes, insufficient documentation and ignored non-functional requirements (NFR). Costs increase easily because cost estimations are based on original application features and implementation plans. It is not possible to estimate discarded or new user stories beforehand. [21]

In waterfall model software implementation work starts after all functionalities are described verbally and visually in requirement specification documentation. While in agile model requirements documentation it is not updated after each version, especially if the software is developed with rapid development cycles. [2] Up to date software requirements documentation is necessary if you need to have new developers added to the development team. The further development of the application is more difficult if the requirements are insufficiently documented. [21]

In the agile model, the requirements engineering activities are often left to the product owner. Backlog type functionality lists are not a sufficient way of creating and managing software requirements. Kauppinen [23] recommends customers, users, management, and technical personnel to be involved in the requirements engineering process. [23]

4.3 Abuse Cases

In Agile development model threat modelling and attack surface recognition is implemented by using security stories and abuse cases. Abuse or misuse cases are often used because for developers it is quite normal to create security-related stories to the iteration backlog. Abuse cases' impact to security is significantly higher than other design phase activities. [15]

Abuse case development is easy to implement, and the impact to the application security is well known. Abuse cases development usually starts with "*white board*" brainstorming session where development, operation and security teams share knowledge and discuss how an attacker tries to abuse the system's possible vulnerabilities and how does it affect the system. [24]

Abuse cases can be detailed descriptions of how attackers will attempt to insert malicious content to input boxes or try to overflow input buffers. Abuse cases are scenarios to harm the system, and how the system should respond actions. Abuse cases and architectural risks should be explained and documented so that the attackers' possible ways of exploiting the application's weaknesses must be described clearly. Test cases will be created based on abuse cases descriptions. [24]

5 Testing and Validation

Vulnerability regression testing is essential to ensure the application's new version's security. It is possible to use vulnerability testing suite for the regression test implementation. Automatically scheduled vulnerability regression tests are a simple and effective way to prevent application vulnerabilities. Regression tests can be implemented at the same time as functional tests. [25]

Security testing is often the first activity that companies implement to ensure application security. Testing is an effective way to find vulnerabilities in applications. Security can be tested manually and automatically using different techniques and tools. Testing techniques have their limitations, which must be taken into account when evaluating the testing results. [18]

Implementing testing automation is essential, but manual testing is needed alongside automation, because testing automation cannot find all flaws. Automatic tests should be combined with manual testing, and in this way the application security can be ensured cost-effectively. [18]

Application automated testing: [18]

- Static analysis security testing (SAST)
- Dynamic analysis security testing (DAST)
- Fuzz testing
- Software composition analysis
- Network vulnerability scanning
- Validating platform mitigations and configurations

5.1 Security Code Review

Security code review can be operated in the same way as functionality code reviewing. Development organizations usually have regular functionality code

reviews, and this makes it easier to implement code security reviews. Common vulnerabilities are checked but the most important part is to find logic-level vulnerabilities which are not possible to find with scanners or automated tools. Security code review experts should in addition to technical knowledge also have a deep understanding about the functionality context: users, functionality, business impact. [25]

Reviewing the code requires in-depth knowledge of the code, and security experts do not always have sufficient experience in software development and compiling. Just looking at the code doesn't help; the reviewer must know variable purposes and how they are used in application. For this reason, McGraw [24] recommends leaving the code review to the development organization responsibility, especially if source code analysis tools are in use. The exception is security experts with hands-on experience in programming languages and finding vulnerabilities at the code level. [24]

Evaluating a flaw is almost impossible without understanding the context of the observation. A human can classify and evaluate risks related to perception more precisely.

5.2 Security Automation, Source Code Analysis

Automated vulnerabilities discovering techniques find security defects from code, but automated techniques are not good at finding logical vulnerabilities. The most used security automation forms are static code analysis, dynamic analysis and vulnerability regression testing. [25]

Static analysers use scripts to evaluate the source code for common mistakes and syntax errors. Static analysis can be carried out at a local development stage, source code repository, and when the code is passed to the master branch. Static analysis tools do not execute code. It is possible to configure the static analysis tool so that it forces developers to use the best practices in coding. [25]

Static analysis is an essential and useful tool, but it also reports false positive flaws, and this can be very frustrating. Static analyser tools find common vulnerabilities and misconfigurations from statically typed program languages, but it is not good for advantaged or chained vulnerabilities discovery. Dynamically typed program languages are difficult to analyse with static analysis tools because variables are dynamically changeable objects and application state is not understandable. [25]

Static code analysis tools detect following vulnerabilities: General XSS, Reflected XSS, DOM XSS, SQL Injections, CSRF, DOS. [25] Table 6 lists vulnerabilities that can be found with static code analysis.

Table 6. Static analysis detects following vulnerabilities. [25]

Exploit	**Description**
General XSS	DOM manipulation with innerHTML
Reflected XSS	Variables pulled from a URL param
DOM XSS	Specific DOM sinks such as setInterval()
SQL Injection	User-provided strings being used in queries.
CSRF	State-changing GET requests.
DOS	Improperly written regular expressions.

5.3 Dynamic Application Security Testing

Dynamic Code Analysis Testing (DAST) is a programming language-independent testing tool that tests the application's behavior. Dynamic code analysis is especially used in web-based applications testing because it can index interfaces and automatically perform tests in the user interface. [18]

Dynamic analysis inspects code post-execution and therefore dynamic code analysis is much more expensive and time-consuming. Dynamic analysis is

carried out in a pre-production environment with all necessary components. Static analysis finds potential vulnerabilities whereas dynamic code analytics detect actual vulnerabilities and compares them against vulnerabilities and misconfiguration models. Dynamic code analysis also discovers application operation side-effect vulnerabilities such as improperly stored sensitive data or side-channel attacks. Properly configured dynamic analysis tools produce less false positive defects and provide deeper information about your application. [25]

5.4 Penetration Testing

Penetration testing focuses on finding human or process-related failures made in software configuration and deployment. The system network topology, firewall placement and communication protocols are checked during testing. Penetration testing considers previous test results and performed risk analyses. [24]

There are several black-box penetration tools available for testing, but it does not make sense to run hundreds of tests without planning. In parallel manual and automated testing tools, the implementation of penetration testing is used. Security experts create a test plan and decide used tools for test cases execution. However, it is worth using network security scanners such as Nmap and Nessus because there is always a risk doing misconfiguration for the complex networks. [24]

Test cases should be planned to follow realistic service user scenarios. Creating risk-based test scenarios is utilized by security experts who have experience from similar applications, and who have the ability to think such as an attacking party. Documented architectural risks and abuse cases are helpful for generating a prioritized risk-based test scenarios list for testing. [24]

Penetration testing is expensive to implement, so the parts of the software to be tested are often carefully selected. The system is examined in a larger context,

which is why tests can find flaws in the application business logic. The real value of penetration testing is achieved when testing the application in a production environment, and finding possible vulnerabilities related to the technical environment and configuration. [18]

Penetration testing is unscalable and time-consuming stage where testers' deep expertise is mandatory. Large organizations often use their own specialized testers or security testing consultants. [18]

5.5 Network Vulnerability Scanning

Network scanning tools find especially third-party and previously found vulnerabilities. These tools are valuable as long as you remember what vulnerabilities they are designed to find. Particularly applications that are deployed in the SaaS environment and operating system environment security is a critical part of the solution. [18]

Network vulnerability testing aims to find vulnerabilities in the network and infrastructure used by the application. To compare the results, the scan should first be performed before application installation, and the baseline results are compared to the situation after the application is installed. In this way, vulnerabilities caused by the application installation can be found. [18]

6 Design Guidelines and Security Practices

Building Security in Maturity Model (BSIMM) organization can evaluate application security maturity level and compare their status to world-class companies. BSIMM is a robust community where participants share information about the best practices. [11]

6.1 Design Guidelines for Secure Systems Engineering

General design guidelines are widely applicable for designing secure systems. Guidelines increase software developers' awareness of security issues, and they can be used as a system validation process checklist. [2] Reference [2] also pointed out that general guidelines are often too indeterminate and cannot be used as such in the application design process, therefore more specific 10 design guidelines for secure systems engineering have been generated, which are based on the following publications Schneier 2000, Viega and McGraw 2001 and Wheeler 2004. [2]

10 design guidelines for secure system:

1. **Base security decisions on an explicit security policy:** Organization security policy is a collection of high-level statements and fundamental security conditions for the company and can be used as a framework for designers. Design decisions should be made and validated based on an organization security policy framework. [2]

2. **Use defence in depth:** Ensure security with several mechanisms. For instance, multifactor authentication is one example of the defence in depth solution where user authentication is a username password combination and authentication is also confirmed with a verification code or pre-registered questions. [2]

3. **Fail securely:** Failures' fallback procedures must be on the same secure level as the system itself. Ensure data encryption and protection so that unauthorized users are not able to read data. [2]

4. **Balance security and usability:** Systems usually become difficult to use if all possible security features are used. Poor usability might lead to situation where users for instance write their passwords on paper and an attacker inside the company can benefit from this situation. [2]

5. **Log user actions:** Collect system user operations' logs at a sufficient level because this prevents e.g., company insiders' attacks and gives tools for analysing attackers actions on the system. [2]

6. **Use redundancy and diversity to reduce risk:** Ensure enough copies of the system data. Avoid placing different system components on the same platform or avoid using the same technology in all parts of the system. [2]

7. **Specify the format of system inputs:** Specify expected system inputs with limitations of input length, character type or forbidden characters. Ensure that unexpected inputs do not cause system failures. [2]

8. **Compartmentalize your assets:** Restrict users to accessing only the information they need for a work assignment. This distinction effectively restricts what information an attacker can access if credentials have been stolen. [2]

9. **Design for deployment:** Ensure the reliability of the deployment process because security problems are usually related to configuration and deployment errors. [2]

10. **Design for recovery:** Ensure system recoverability in a situation where the system has been attacked and data has been leaked.

Prepare for the situation where users must update their passwords before the system can be used again. [2]

6.2 BSIMM Best Practice Collection

The BSIMM model is a collection of the best practices of information security. It can be used for measuring security maturity, tracking current security activities, and defining organization final maturity level. [11] BSIMM12 software security framework includes 12 practices and 122 activities. The framework practices are organized under four high-level domains: Governance, Intelligence, SSDL Touchpoints and Deployment. [26] Table 5 presents the BSIMM framework's four domains and 12 practice categories.

Governance related activities consists of practices which help to manage, organize, and measure a software's security. It also includes company personnel competence development activities. Governance level activities are organization security foundations, and they are managed by the organization's management. [26]

Intelligence is an organization-level knowledge collection that consists of applicable security standards, proactive security guidance, available security practices and threat modelling. [26]

SSDLTouchpoints include software development process assurance activities that are performed before the application is taken in use in a production environment. Activities also include software security methodologies. [26]

Deployment includes Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security. [26]

Table 5. BSIMM framework four activity domains with 12 practices. [26]

Covernance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy & Metrics (SM)	Attack Models (AM)	Architecture Analysis (AA)	Penetration Testing (PT)
Compliance & Policy (CP)	Security Features & Design (SFD)	Code Review (CR)	Software Environment (SE)
Training (T)	Standards & Requirements (SR)	Security Testing (ST)	Configuration Management & Vulnerability Management (CMVM)

Past 15 years "*Shift left*" movement mindset encouraged to move security testing activities early stage in the development process. Nowadays "*Shift everywhere*" approach encourage to implement automated security tests application pipeline so testing is continuous throughout the application lifecycle. Automated tests can for instance verify and monitor continuously Application Programming Interfaces (API) received traffic. Continuous defect discovery importance has grown in latest BSIMM13 report. Code Review and Security Testing practices efforts almost doubled compared to Penetration Testing and Architecture Analysis practices efforts. [28]

Based on the BSIMM13 report data it is recommendable to consider following actions:

1. Take in use automated static and dynamic testing tools to secure application. Security tools help to identify vulnerabilities and fixing found flaws. Tools are useful with in-house developed applications, third-party and open-source applications. [28]

2. Collect security testing tools data and use for company security policies enforcement. Combined data can be used for driving systems development life cycle (SDLC) and governance process improvements. [28]

3. Take in use automated security testing. Use repeatable effective automation and give up manual activities. [28]

4. Take in use automated inspections in SDLC process. Use pipeline-driven testing technologies and give up manual penetration testing and code reviews. [28]

5. Take in use software bill of materials (SBOM) inventory tool for listing open source and third-party components. Automated BOM tools listing used components vulnerability information, versions, and licenses. This simplifies open-source components dependencies tracking. [28]

6.3 Three Categories for Security Practices

Reference [27] divides security practices into the following vulnerability related categories, which are prevention, detection, and response. Detection category security practices are more reactive actions than prevent category practices. Vulnerability response category practices take place after software is deployed to the production environment. Once the application is installed to the production environment, only the response category security practices are in use. Organizations would benefit tremendously if proactive prevention and detection category security practices are put in place to the application development process. [27]

The development team strives to prevent vulnerabilities from occurring, so it is good to provide developers all necessary software security tools to help finding implementation bugs or design flaws early as possible. Of course, all developers and other stakeholders need to get training to exploit security practices in a secure system design and development. Automated static code analysis tools must be easy to use and must be a part of the developer’s current workflow. [27]

Table 7. Categorized security practices. [27]

Practice	**Category**
Use abuse cases and threat models	Vulnerability Prevention
Use secure coding standards	Vulnerability Prevention
Security policies and regulations	Vulnerability Prevention
Use manual and automated code reviews	Vulnerability Detection
Ensure security tests	Vulnerability Detection
Use external penetration testers	Vulnerability Detection
Create incident responses	Vulnerability Response
Use application behavior monitoring	Vulnerability Response
Fix software bugs	Vulnerability Response

6.4 Best Practices in Software Security

Reference [24] described best practices for improving application security (Figure 11). The activities outlined below are worth implementing at the various stages of the development process, which include the definition, design, testing, code review, acceptance testing, and deployment phases. Security experts usually only have reactive tools to notice vulnerabilities, and therefore it is important to include proactive notice activities to the application development process. Bad or erroneous solutions made at the design stage are almost impossible to correct later without a lot of effort. [24]

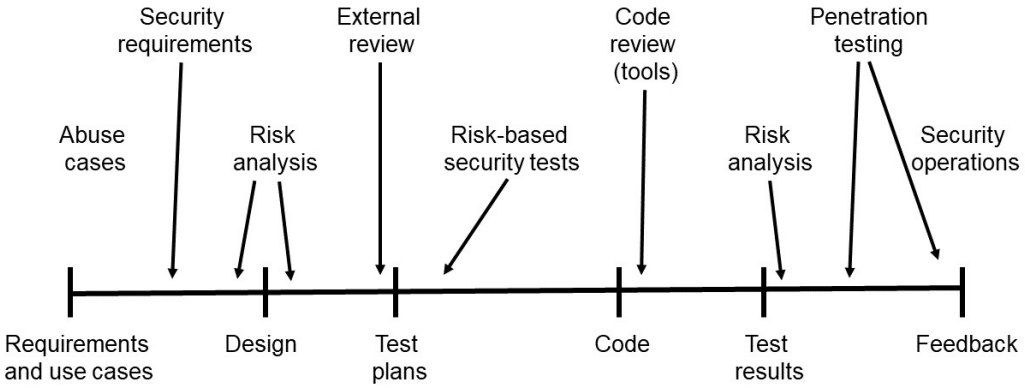


Figure 11. Software security practices mapped onto the development process stages. [24]

Table 8. Security practices in development process phases. [24]

Development phase	**Practice**
Requirement phase	In the software development process requirement phase it is recommendable to try to build possible abuse cases and based on those useful security requirements cases can be created. Security professionals and developers work together to create scenarios for abuse cases. Use cases can be based on real life examples, but it is important not to overstate them and stick to the facts. [24]
Design phase	Risk analysis should be made from business and architectural points of views. From a business perspective, it is important to outline how security-related incidents affect the misused system and what the concrete consequences for the company are. The analysis from an architectural perspective includes an assessment of technical security risks and their comparison to business impacts. [24]
Test planning phase	Risk-based testing simulates step-by-step attacker actions for breaking the target system. Test cases should be similar with the real user scenarios. Information security professionals can help to create test cases based on their knowledge of real security issues. [24] Security features testing should be done by experts who understand more about the software security protection methods such as encryption, user

	identification, logs, confidentiality, and authentication. Risk-based test scenarios are used to assess and prioritize risks related to the application architecture. [24]
Code review phase	The software code is checked with a static analysis tool and by another developer, if no bugs are found in the code it can be installed for testing. Security bugs are attempted to be found automatically before the software's release. [24]
Penetration testing phase	Penetration testing phase inspection is focused to find out human and procedural related failures. Penetration test cases are designed to ensure that previously identified risks or vulnerabilities cannot be exploited. Network security scanners for example Nessus are very useful because there are many ways to do misconfigurations to the system networks. Application security test scanners are not reliable enough, so it is necessary that skilful professionals do manual testing for the software. [24]
Deployment and operations	Software deployment environment carefully implemented configuration and customization guarantees a high level of security for the system. Designing the application's operating environment requires that the network components, operating system, and application security configuration and settings be implemented as planned. [24]

7 Improvements to the Application Development Process

The thesis objective is to improve the security control management of case company application development and operation process. The DSRM research method objective centered solution approach was chosen because the case company has started to adopt the DevOps process model for application development and operation, and they want to improve application development process security management in a situation where application release cycle is accelerating.

7.1 The Company's Current Development Process

The company used in the case study operates in the energy industry. The company's systems are usually acquired as services according to the SaaS model contracts, which includes system maintenance and continuous improvement development. Web applications development is not the company’s core business, and the company does not have its own application development unit. Service design and application development work is purchased from several application development companies. Consultants work case company premises or remotely. The company has its own architects, product owners and system operation managers. The development teams use basic agile development model practices extended with DevOps method practices, structure to work gives scrum ceremonies and Scrumban is used for task management.

The case company uses test automation and static code analysis tools. Developers have used static code analysis tools in their development environments. Using the generic static code analysis tool enables automating code reviews, preventing to release incomplete code and sharing status reports.

7.1.1 DevOps Maturity Level Evaluation

Organization DevOps method maturity level was evaluated based on Eficode's DevOps method maturity model framework. Company DevOps method maturity level was quite high 24,5 points from total of 28 points. Table 9 shows company DevOps maturity level evaluation results in 7 categories.

Table 9. Company DevOps maturity level evaluation. [10]

Evaluation area	Maturity level description	Points
Leadership	Development projects can be started quite flexibly. There is a clear strategic control method for making decisions. Achieved goals reported every week. [10]	3,5
Organization and culture	The teams communicate with each other regularly and develop working methods together. Service maintenance is part of the team. [10]	4
Environments and releasing	Releasing can be done continuously and automatically. Recovery processes work as expected. [10]	4
Continuous integration	Integration covers the whole product and is connected to acceptance testing. Dependencies are known and partly controlled. [10]	3
Quality assurance	Acceptance tests present the system's requirements and guide the system's development work. [10]	3
Visibility and reporting	Requirements status can be monitored in real time tests and released features. Metrics are collected and used for improving process. [10]	3,5
Technology and architecture	Technology is new and well-supported. System functions can be partly performed via interfaces. The architecture and technology choices are appropriate for the purpose. [10]	3,5

To raise case company DevOps method maturity level making the start of development projects easier, speeding up the feedback cycle, improving

visibility using and expanding metrics collection are key. Dependencies are known, but their management needs to be improved. [10]

In Figure 12 is shown company DevOps operation model maturity level based on Eficode's document.

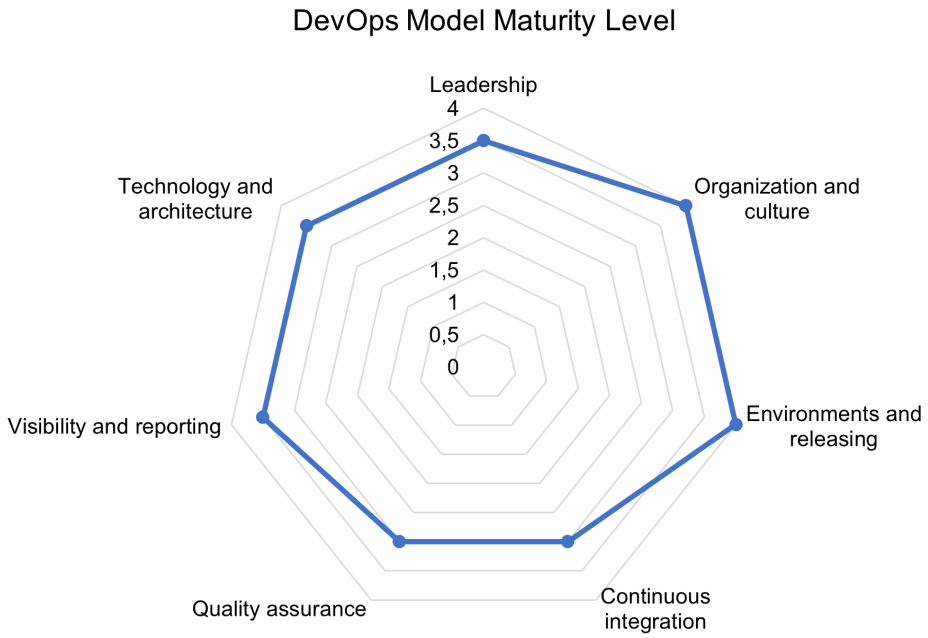


Figure 12. Company DevOps model maturity level visualized.

7.1.2 BSIMM Maturity Level Evaluation

The BSIMM provides a starting point for comparing high-level activities with the participating 128 companies used practices. Activity levels indicate how often activities are used in companies. Frequently observed activities are classified as level 1, and levels 2 and 3 are less commonly used activities. [26]

Case company software security maturity level was evaluated by using the BSIMM12 framework baseline information about the other companies' security activities and comparing them with used security activities in case company. Security practice areas Strategy & Metrics, Standards & Requirements and Configuration & Vulnerability Management were the only practice areas where case company was slightly below the level of all companies in comparison, but

Security Testing area practices was significantly lower level than other companies. In Figure 13 is shown case company and all firms security maturity level. In Figure security practice areas are categorized with prevention, detection and response categories.

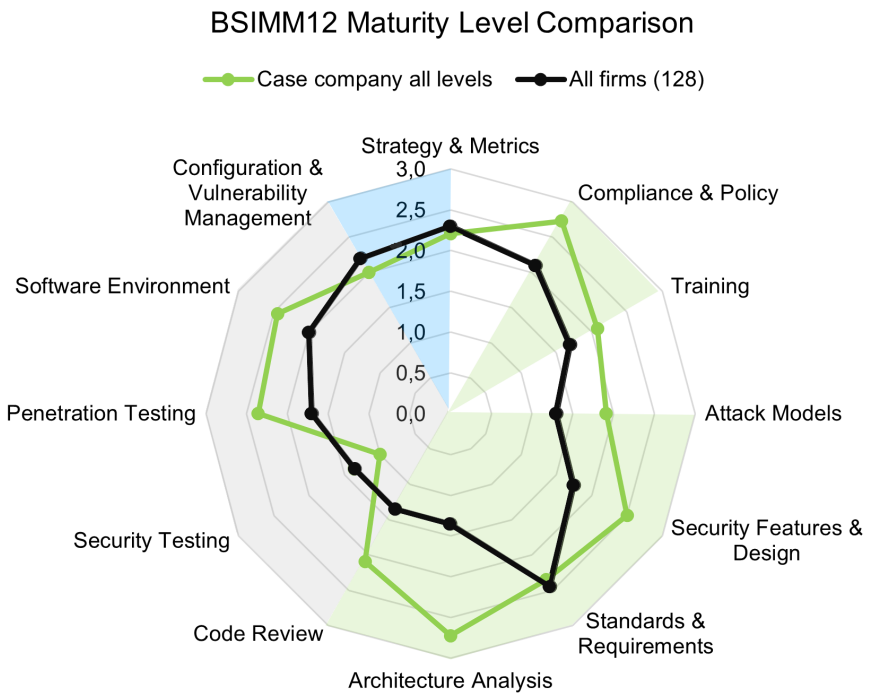


Figure 13. Case company software security maturity level comparison with all BSIMM12 framework firms. Security practices prevention category practices is marked green color, detection category practices are marked grey and response category practice is marked blue color.

According to the BSIMM12 framework evaluation the case company's best maturity level was in practice areas Compliance & Policy, Security Features & Design, Architecture Analysis and Penetration Testing. Strategy & Metrics, Training, Standards & Requirements, Code Review, Software Environment and Configuration & Vulnerability Managements practice areas can be improved. Attack Models and Security Testing practice area security most effective activities must be implemented, especially because development and operation teams working methods are moving towards the DevOps model. Figure 14 showcases company maturity level points enriched with activities levels 1-3 situation.

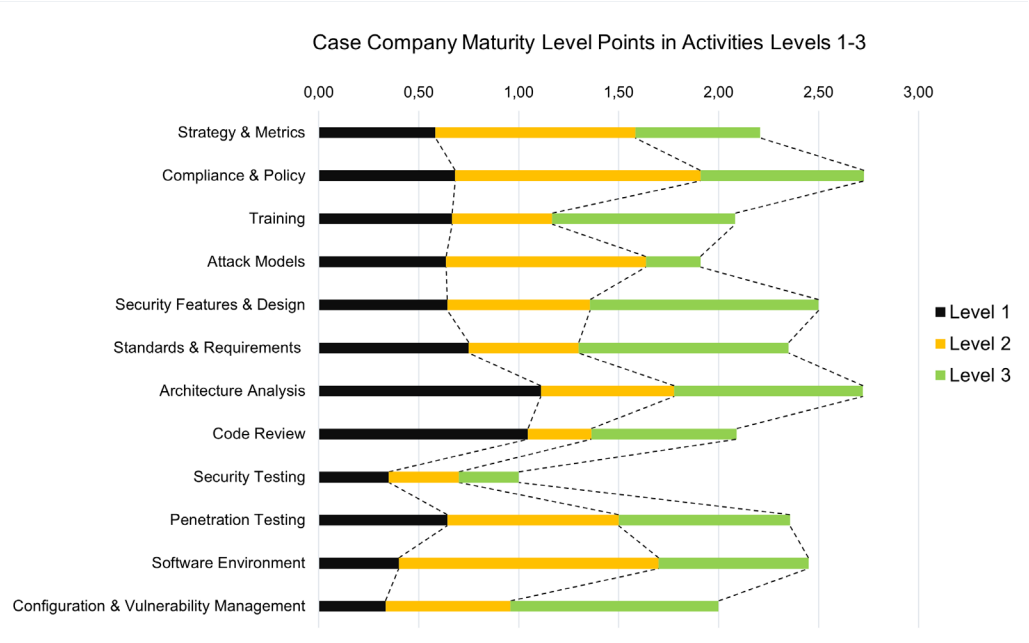


Figure 14. Case company maturity level activities in 1 to 3 level. Attack Models and Security Testing area practice activities has great potential to increase the maturity level.

The Attack Models' practice area activities implementation needs to be improved, especially 2 and 3 levels activities. All area activities belong to the prevention category security procedures, and they are carried out in the development process design phase. Attack modelling includes threat model creation, abuse case development and increasing deeper knowledge of attack methods [29]. Data classification and inventory has been implemented in the organization.

The Attack Models' practice area second level activities are not fully used in organization. Attack patterns and abuse cases development is partly in use but collected knowledge should be shared with service developers and used to accumulate attack intelligence. Attack knowledge should be shared at least in the software developer's internal forum. Understanding attackers' motivation and capabilities is important part of prevention category activity. Identifying attackers is sometimes difficult because the attacker can theoretically be a part of the company's personnel or a consultant. [29]

The Attack Models' practice area third level activities received the least points. The organization should emulate attackers' actions with automation and monitor automated resource creation. Mimicking attackers' actions needs customized tools to consider infrastructures, technologies, and configurations in use. DevOps development method allows tools embedding into pipelines and automation. Succeeding in monitoring requires external web crawling, uses of orchestration and virtualization metadata, and cloud service provider API endpoints querying. [29]

The Attack Models' practice area maturity level was 21 points from total of 36 points. Average was 1,91 from total of 3 points. Table 10 shows Attack Models practice area 11 activities evaluation results in 3 levels.

Table 10. Attack Models practice area activities evaluation. [26]

Activity	Level	Priority
Create a data classification scheme and inventory [26]	1	3
Identify potential attackers [26]	1	2
Gather and use attack intelligence [26]	1	2
Build attack patterns and abuse cases tied to potential attackers [26]	2	2
Create technology-specific attack patterns [26]	2	2
Maintain and use a top N possible attacks list [26]	2	2
Collect and publish attack stories [26]	2	2
Build an internal forum to discuss attacks. [26]	2	3
Have a research group that develops new attack methods [26]	3	1
Create and use automation to mimic attackers [26]	3	1,5
Monitor automated asset creation [26]	3	0,5

Security testing practice area activities implementation needs to be improved, on all tree levels of activities. All area activities belong to the detection category security procedures, and they are carried out in the development process verification phase. Security Testing practice area maturity level was 10 points from total of 30 points. Average was 1,00 from total of 3 points. Table 11 shows Security Testing practice area 10 activities evaluation results in 3 levels.

Table 11. Security Testing practice area activities evaluation. [26]

Activity	Level	Points
Ensure QA performs boundary value condition testing [26]	1	0,5
Drive tests with security requirements and security features [26]	1	1,5
Integrate opaque-box security tools into the QA process [26]	1	1,5
Share security results with QA [26]	2	1,5
Include security tests in QA automation [10]	2	1,5
Perform fuzz testing customized to application APIs [26]	2	0,5
Drive tests with risk analysis results [26]	3	0,5
Leverage coverage analysis [26]	3	0,5
Begin to build and apply adversarial security tests (abuse cases) [26]	3	1,5
Implement event-driven security testing in automation [26]	3	0,5

Organization should share security testing for example threat modelling, code reviews, penetration testing and composition analysis results with all relevant teams. Security flaws create a basis for discussing attack models and patterns. Automated GitHub pipeline testing reports create a basis for deeper conversation with stakeholders. Another area for improvement is to implement a

code coverage analysis which increase depth of security testing. It is important to know how widely test cases use code. This is a different issue than security requirements coverage. The third improvement area is to use risk profiles for creating test cases. Architecture analysis and design reviews' results can lead creating tests. High-risk defects can be found in adversarial tests based on a risk profiles and shared security defects data. [29]

Security testing includes boundary value condition, opaque-box, event-driven, fuzz, and adversarial security testing automation activities. [29] Table 12 describes 7 different security testing types that should be implemented to the organization development process.

Table 12. Security testing types with explanations. [29]

Activity	Description
Boundary values testing	Tests using unexpected inputs such as boundary values and exploring application operation to find weaknesses. [29]
Security features testing	Security features can be tested in the same way that other functionalities. Administrative accounts' security mechanisms for example lockout, transaction limits, entitlements must be tested with expected and unexpected inputs. [29]
Opaque-box testing tools	Opaque-box testing tools (e.g. web application scanners) can be integrated into the development process with internal cloud-based toolchains. [29]
Automated security tests	Security tests should be automated and running beside performance, functional and other tests. Automated tests can be provoked manually or automatically as a part of pipeline processes, outside pipeline performed manual tests not providing feedback on time. [29]
Fuzz testing	Fuzzing tests include built-in intelligence from application business logic and API interfaces. [29]

Adversarial testing	Testers moving to use attacker's point of view to try to break the system with abuse cases-based test cases. Security policies, attack intelligence, standards, and N attack lists create a basis for abuse and misuse cases creation. [29]
Event-driven testing	Event-driven testing is automation implemented to the pipeline that is triggered when new code is put in a repository. Automation activates testing after defined conditions are reached. [29]

Penetration Testing practice area activities are quite well taken into use, but level 1 has two activities that need to be improved. All area activities belong to the detection category security procedures, and they are carried out in the development process verification phase. Penetration Testing practice area 7 activities maturity level was 16,5 points from total of 21 points. Average was 2,36 from total of 3 points.

Penetration testing reports are shared with development and operations experts. In addition, DevOps team can follow other testing activities results real-time in social tools such as JIRA and Slack. Change requests should be directed to the development team more quickly, although requests are also a part of vulnerability management process. This process improves security and highlights importance of security vulnerabilities fixing. [29]

Another improvement target for the practice area is to implement internal penetration testing tools to an available development process. Automated tools complete manual testing and improve testing process efficiency. There are products for application and network penetration testing. [29]

Service operational monitoring can identify defects from the production environment, and all found defects must be fed into the development team backlog. Production environment defects enable learning and change behaviour in the organization. Defects’ proper root-cause analysis exposes at which stage of the application life cycle error has slipped to the production environment.

Operational experiences can lead SSDL process changes. Service infrastructure automated security verification operations are mandatory to implement. [29]

Code review can be set as mandatory, so that for example unacceptable review results stop the release process automatically. Secure coding standards violation can cause code rejection and move code back to the development for fixing. [29]

7.2 Improvement Proposal

Only a few companies reported using automated tests to detect vulnerabilities, even though automation can detect design and implementation errors. When implementing security tests, it is important to pay attention to the fact that they do not cause disruptions to the current development workflow. Penetration testing is usually done late in software development, and this can cause unwanted delays. Penetration testing should be automated to the highest extent possible. [27]

Companies usually first adopt practices related to responding to security deviations, then they move on to detection practices, and only lastly do preventive practices for ensuring product security. Proactive practices are the most recommendable, so that you don't end up just reacting to security violations and thus cause unnecessary harm to service users and company. [27]

In addition, software developers must be trained to use security tools and practices for application technical planning and implementation. Application development must also consider malicious actions taken by attackers, unintentional mistakes made by well-intentioned users, such as clicking on harmful links, and protect users from their own actions. [27]

Cooperation between operations, developers and security specialist is key to improve application security. If developers consider security experts to be barriers to development work, it will have a significant impact on success. The security expert must be more of an enabler of development work so that teams can find solutions to improve the application and system security. Developers accept guidance and advice from security experts, but it must happen in a good spirit of cooperation. [24]

In DevOps development and operation model activities is recommendable to add several improvements for security practice areas such as Attack Models, Standards & Requirements, Code Review, Security Testing, Penetration Testing and Configuration & Vulnerability Management. In Figure 15 is the yellow color marks the areas where there is potential to make improvements, especially by improving the automation level in the process.

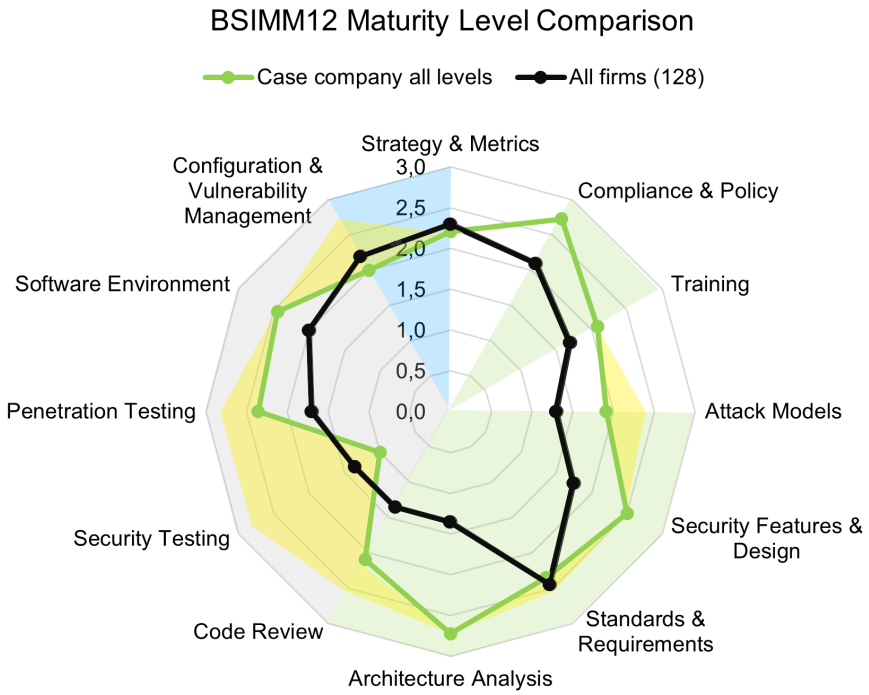


Figure 15. Case company security maturity level comparison with all BSIMM12 framework firms. Security practices prevention category practices is marked green color, detection category practices are marked grey and response category practice is marked blue color. Yellow color indicates the practice areas which especially needs improvements.

Security activities were selected based on the BSIMM reports and other references recommendations. Case company software security maturity level was evaluated with BSIMM12 framework, which helped to pick out the activities that need the most improvement and where there is the most potential for improving security. Table 13. Lists all security activities that require implementation or improvements.

Table 13. Security activities require implementation or improvements. [26]

Requirements phase	
	1. Identify open-source components 2. Control open-source risk
Design phase	
	1. Identify potential attackers 2. Use attack intelligence 3. Build attack patterns and abuse cases tied to potential attackers 4. Create technology-specific attack patterns 5. Use a top N possible attacks list 6. Collect and publish attack stories 7. Use automation to imitate attackers 8. Monitor automated asset creation 9. Ensure proper communication
Development phase	
	1. Make code review mandatory 2. Use automated code review tools with tailored rules 3. Enforce coding standards 4. Ensure proper communication
Verification phase	
	1. Automated security tests 2. Opaque-box testing tools 3. Use security features, event-driven, boundary values, adversarial and fuzz testing

	4. Code coverage analysis 5. Share security test results 6. Use penetration testing tools internally 7. Feed test results to the defect management 8. Control open-source risk
Release phase	
	1. Use penetration testing tools internally
Operation phase	
	2. Identify defects and feed them back to development 3. Prevent software bugs found in operations 4. Automate operational infrastructure security verification 5. Identify open-source components 6. Control open-source risk 7. Ensure proper communication

The company current application development and operation processes is shown in Figure 16, and it includes the first proposed improvements to the security activities.

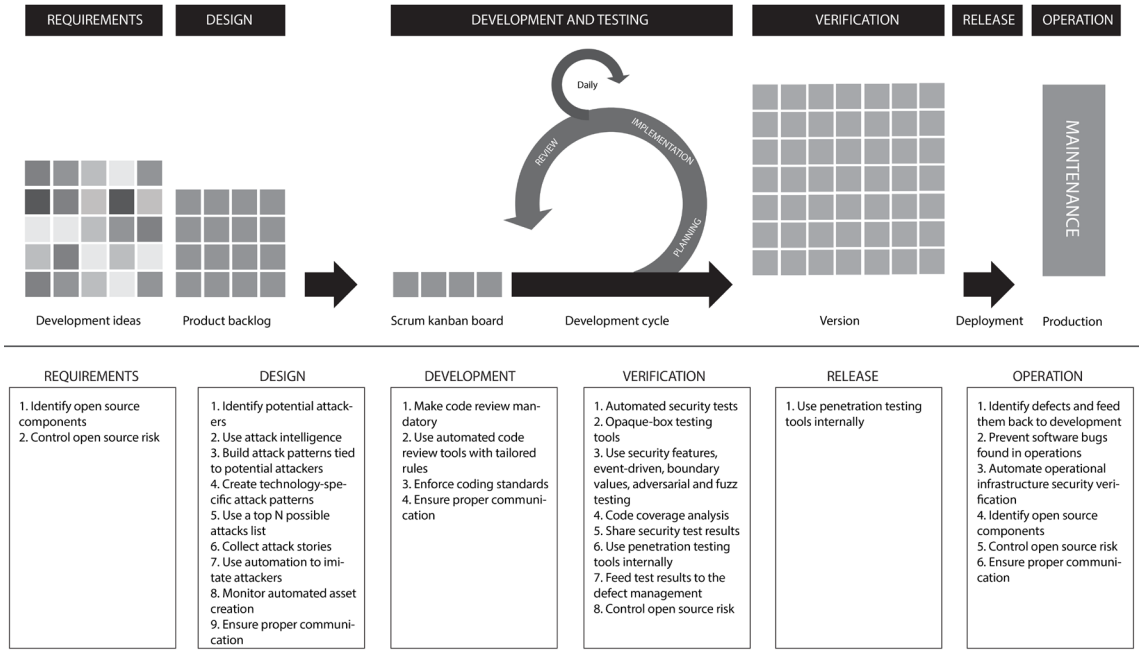


Figure 16. The first case company application development and operation processes with stages and recommended security activities.

7.3 Final Proposal

The case company cybersecurity manager pointed out that the activities in the Attack Model practice area are already at a good level based on the evaluation results and do not require urgent actions. Based on this consideration, it was decided to classify the selected measures by the score they received in the evaluation. The activities were classified as follows: 0-1 points = high, 1<x<2 points = medium, 2-3 points = low.

All 28 selected security improvement activities were classified by their implementation urgency. Based on the classification, there were 28.6% high level activities, 39.3% medium level activities and 32.1% low level activities. Over 57 % of the recommended activities can be implemented by expanding automation. Manual process improvements are required in 43 % of all selected activities. It is recommended to implement at least 8 high-level and 12 medium-level activities. Almost 75 % of selected activities are implementable with automation procedures or tools and 25 % of selected activities needs manual procedures development. Figure 17 shows the selected practice area security activities classified by implementation urgency.

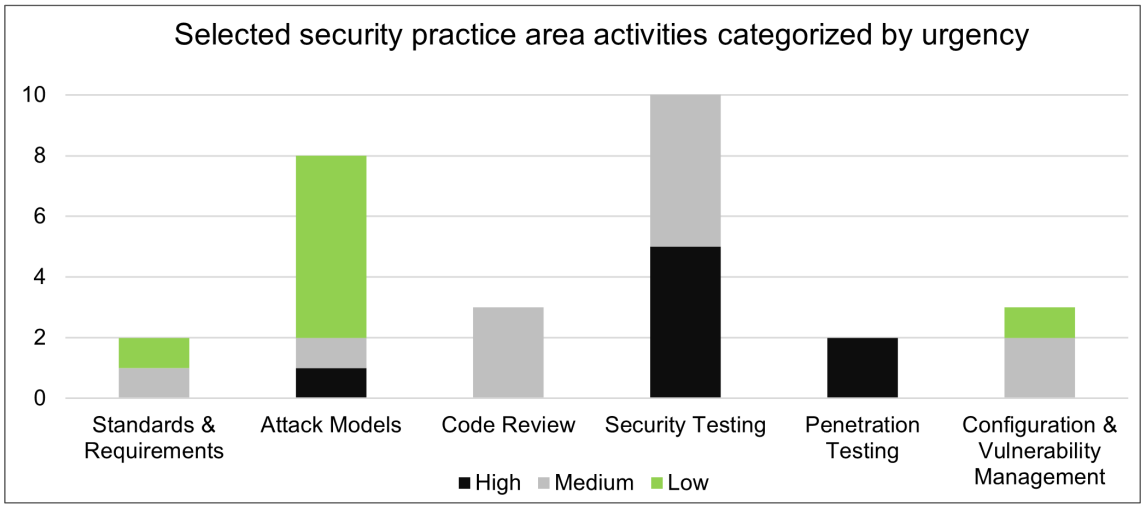


Figure 17. Selected security practice area activities classified by them implementation urgency.

Table 14 presents all selected activities categorized by urgency and automatability. In appendix 1 is listed all proposed activities require implementation or improvements.

Table 14. Selected activities categorized by urgency and automatability.

Activity [26]	**Urgency**	**Activity type [14]**
Identify open source components	Medium	Automation
Control open source risk	Low	Automation
Identify potential attackers	Low	Manual
Use attack intelligence	Low	Manual
Build attack patterns and abuse cases tied to potential attackers	Low	Manual
Create technology-specific attack patterns	Low	Manual
Use a top N possible attacks list	Low	Manual
Collect and publish attack stories	Low	Manual
Use automation to imitate attackers	Medium	Automation
Monitor automated asset creation	High	Automation
Make code review mandatory	Medium	Automation
Use automated tools with tailored rules	Medium	Automation
Enforce coding standards	Medium	Automation
Ensure QA performs boundary value condition testing	High	Automation
Drive tests with security requirements and security features	Medium	Automation
Integrate opaque-box security tools into the process	Medium	Automation
Share security results with team	Medium	Manual
Include security tests in automation	Medium	Automation

Perform fuzz testing customized to application APIs	High	Automation
Drive tests with risk analysis results	High	Manual
Leverage coverage analysis	High	Manual
Begin to build and apply adversarial security tests	Medium	Manual
Implement event-driven security testing in automation	High	Automation
Use penetration testing tools internally	High	Manual
Feed results to the defect management system	High	Automation
Identify software defects found in operations monitoring and feed them back to development	Medium	Automation
Prevent software bugs found in operations	Low	Manual
Automate operational infrastructure security verification	Medium	Automation

This thesis does not include precise descriptions on how activity can be implemented or improved. Figure 18 presents a case company high-level software development and operation processes with added final security activities improvements proposal. Application development and operation process with activities requiring improvements most urgently is on appendix 2.

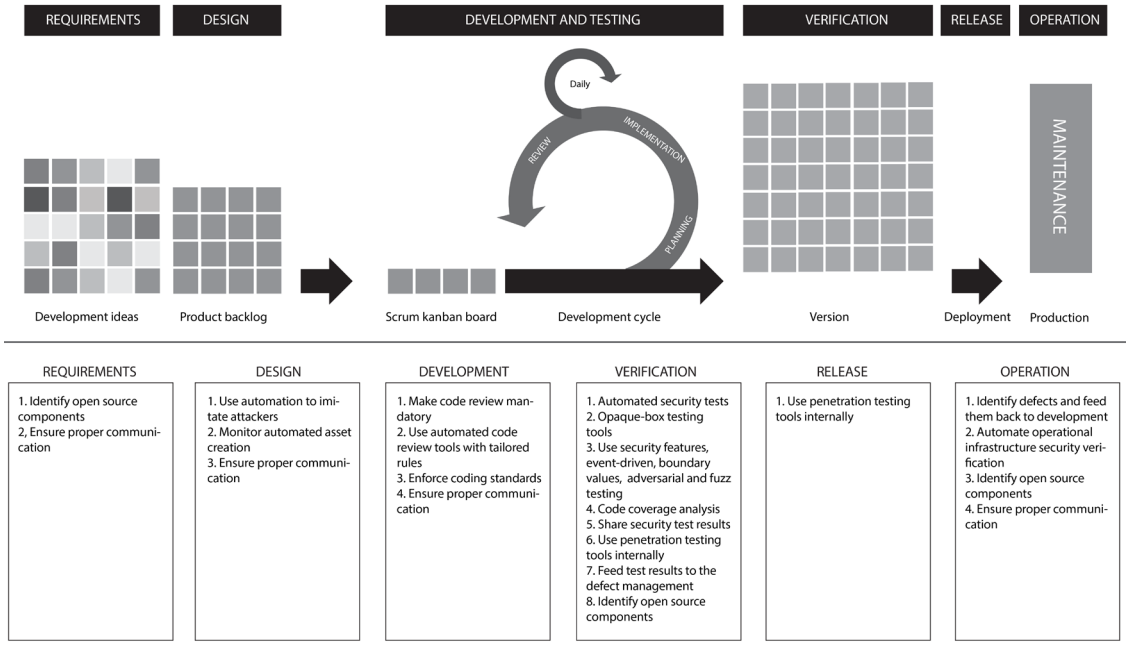


Figure 18. Case company application development and operation processes with stages and recommended security activities.

8 Discussion and Conclusions

The web applications are freely available on the Internet and are thus constantly exposed to various attacks. There are various parties on the Internet that actively and automatically search for vulnerabilities in services. Web applications administrators strive to develop different control and monitoring methods for their services, and despite this, the services could be attacked at any time. It is mandatory to implement a Web Application Firewall (WAF) and shield solutions to protect web applications and detect potential attacks.

Adapting principles, guidelines, and practices to the various stages of the Agile development model process is essential to prevent attacks and strengthen web application security. In addition to these, it is important to include automated security testing in the service development, deployment, and maintenance processes. However, not all vulnerabilities can be found with security test automation, so in any case, manual penetration testing must be carried out on the service.

In this Thesis study, the main goal was to find suitable security practices to improve application development and operation processes. The case company has noticed difficulties to ensure application security when using Agile development and DevOps processes. Another noted problem was correcting security findings at the end of the development process, which made fixing more difficult and expensive.

The Design Science Research (DSR) was used as a research method in the thesis, because it enables a systematic approach, and it is suitable for information system-related research projects. DSR methodology objective-oriented approach was selected for use in the thesis.

The thesis process was started by reviewing the literature on this subject containing information about software development and operation methods, and

how it would be possible to add security activities and controls that improve security processes.

After the conceptual basis was achieved, the case company's current application development and operational processes were evaluated by using Building Security In Maturity Model (BSIMM) framework.

The case company has tenaciously and systematically improved web application security in past years. Software security maturity level comparison with all BSIMM12 framework 128 firms was very informative and the current situation is quite good. Security Testing practice area security activities maturity level was significantly below than companies used in the comparison. Strategy & Metrics, Standards & Requirements and Configuration & Vulnerability Management practice areas were only slightly below the level of all companies in comparison. Security Testing practices area activities must be implemented, especially because development and operation teams have adopted it as a working method DevOps model.

In the latest BSIMM13 report was pointed out that continuous defect discovery, code review and security testing area practices are the most important areas to do improvements [28]. The latest BSIMM13 report recommendations were considered in the proposal.

BSIMM13 report recommendations:

1. Take in use automated static and dynamic testing tools to secure application. [28]

2. Collect security testing tools data and use for company security policies enforcement. [28]

3. Take in use automated security testing. [28]

4. Take in use automated inspections in SDLC process. [28]

5. Take in use software bill of materials (SBOM) inventory tool for listing open source and third-party components. [28]

The improvement proposals for the development and operation process are based on the observations, research findings presented in the literature and the case company process evaluation results. The selected security practice area 28 activities were classified according to the points they received in the evaluation. Activities with the highest scores were placed in the low-level urgency category, and activities with the lowest scores were placed in the high-level urgency category. High and medium level activities are recommended to be implemented first. Based on the evaluation low-level activities situation is reasonable, so there is no need to improve them as urgently.

The final proposal includes 20 high and medium level security activities. 75% of the proposed security activities can be implemented using automation. 25% of the activities require manual procedures improvements. Security activities were also categorized vulnerability related categories: prevention, detection, and response. Proactive prevention and detection category security practices are most recommendable to use. Proposal includes 15 % prevention category activities, 75 % detection category activities and 10 % response category activities.

At the end of the day, technical solutions are only one part of the work. The key to developing high quality security services is collaboration between developers, security experts, architects and other relevant stakeholders. [20, 24]. Effective collaboration between different roles guarantees better opportunities for success. Managers should find a way to support a confidential atmosphere and cooperation between all essential work groups on the team. It is challenging to build a confidential atmosphere between different stakeholder groups. In the worst-case team members start working against each other, and this typically causes serious conflicts and critical problems for information flow. [21]

References

1 Stolberg S. Enabling Agile Testing through Continuous Integration. Agile Conference. 2009: 369-374.

2 Sommerville I. Software Engineering. 10th edition. Pearson. 2015.

3 Cusumano MA, Yoffie DB. Software Development on Internet Time, Computer. 1999; 32(10): 60–69. Available from: https://ieeexplore.ieee.org/document/796110 DOI: 10.1109/2.796110.

4 Peffers K, Tuunanen T, Gengler CE, Rossi M, Hui W, Virtanen V, Bragge J. The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. 1st International Conference, DESRIST 2006 Proceedings. Claremont Graduate University. 2006; 1: 83-106. Available from: https://jyx.jyu.fi/handle/123456789/63435#.

5 Conboy K, Gleasure R, Cullina E. Agile Design Science Research. New Horizons in Design Science: Broadening the Research Agenda. 2015: 168–180. Available from: https://link.springer.com/chapter/10.1007/978-3-319-18714-3_11 DOI: 10.1007/978-3-319-18714-3_11.

6 Larman C, Basili VR. Iterative and incremental development: A brief history. Computer. 2003; 36(6): 47–56. Available from: https://ieeexplore.ieee.org/document/1204375 DOI: 10.1109/MC.2003.1204375.

7 Lassenius C. Software Processes [web streaming video]. Aalto University; 2016 [cited 2019 Jan 23]. Available from: https://vimeo.com/aaltose/review/116611686/65c23e1d27.

8 Ceschi M, Sillitti A, Succi G, De Panfilis S. Project management in plan-based and agile companies. IEEE Software. 2005; 22 (3): 21-27.

9 Lassenius C. Software Project Management [web streaming video]. Aalto University; 2015 [cited 2019 Mar 24]. Available from: https://vimeo.com/aaltose/review/122133092/9b1cd21ac8.

10 Hämäläinen H, Virkkala R, Klemetti M, Jokiniemi K. Pikaopas, DevOps. Eficode Oy, 2016 [cited 2022 Aug 27]. Available from: https://www.eficode.com/learn/devops-opas.

11 Koskinen A. DevSecOps: building security into the core of DevOps. University of Jyväskylä. 2019. Available from: https://jyx.jyu.fi/handle/123456789/67345

12 Lwakatare L. E, Kilamo T, KarvonenT, Sauvola T, Heikkilä V, Itkonen J, Kuvaja P, Mikkonen T, Oivo M, Lassenius C. DevOps in practice: A multiple case study of five companies. Information and Software

Technology. 2019; 114: 217-230. Available from: https://doi.org/10.1016/j.infsof.2019.06.010.

13 Akbar MA, Smolander K, Mahmood S, Alsanad A. Toward successful DevSecOps in software development organizations: A decision-making framework. Elsevier, Information and Software Technology. 2022; 147. Available from: https://doi.org/10.1016/j.infsof.2022.106894.

14 Rahman AA, Williams L. Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices. IEEE/ACM International Workshop on Continuous Software Evolution and Delivery, CSED. 2016: 70−76.

15 Rindell K, Ruohonen J, Holvitie J, Hyrynsalmi S, Leppänen V. Security in agile software development: A practitioner survey. Information and Software Technology. 2021; 131. Available from: https://doi.org/10.1016/j.infsof.2020.106488.

16 Lassenius C. Software Evolution and Configuration Management [web streaming video]. Aalto University; 2016 [cited 2019 Mar 17]. Available from: https://vimeo.com/121640701

17 Engblom C. Podcast: Agile transformation at Ericsson Finland [web streaming video]. Aalto University; 2013 [cited 2019 Feb 2]. Available from: https://lassenius.files.wordpress.com/2013/01/engblom_mastered.mp3.

18 SafeCode. Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program. Software Assurance Forum for Excellene in Code. 3rd edition. 2018.

19 Männistö T. Podcast: Software Architecture Basics [web streaming video]. Aalto University; 2013 [cited 2019 Feb 17]. Available from: https://mycourses.aalto.fi/pluginfile.php/884704/mod_resource/content/11/Mannisto.mp3.

20 Abrahamsson P, Babar MA, Kruhten P. Agility and Architecture: Can They Coexist? IEEE Software. 2010; 27 (2): 16-22.

21 Cao L, Ramesh B. Agile Requirements Engineering Practices: An Empirical Study, IEEE Software. 2008; 25 (1): 60-67.

22 Nuseibeh B, Easterbrook S. Requirements Engineering: A Roadmap. Proceedings of the Conference on the Future of Software Engineering. ACM Press. 2000: 35-46.

23 Kauppinen M. Podcast: Requirements engineering [web streaming video]. Aalto University; 2013 [cited 2019 Feb 10]. Available from: https://mycourses.aalto.fi/pluginfile.php/884690/mod_resource/content/11/Kauppinen.mp3.

24 McGraw G, Van Wyk KR. Bridging the Gap between Software Development and Information Security. IEEE Security & Privacy. 2005; 3(5): 75-79.

25 Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. 1st edition. O'Reilly. 2020.

26 Erlikhman E, Ewers J, Migues S, Nassery K. Building Security in Maturity Model (BSIMM). Trends & Insights Report. Version 12. 2021.

27 Williams L, McGraw G, Migues S. Engineering Security Vulnerability Prevention, Detection, and Response. IEEE Software. 2018; 35(5): 76-80.

28 Boote J, Erlikhman E, Gardner S, Migues S. Building Security in Maturity Model (BSIMM). Trends & Insights Report. Version 13. 2022.

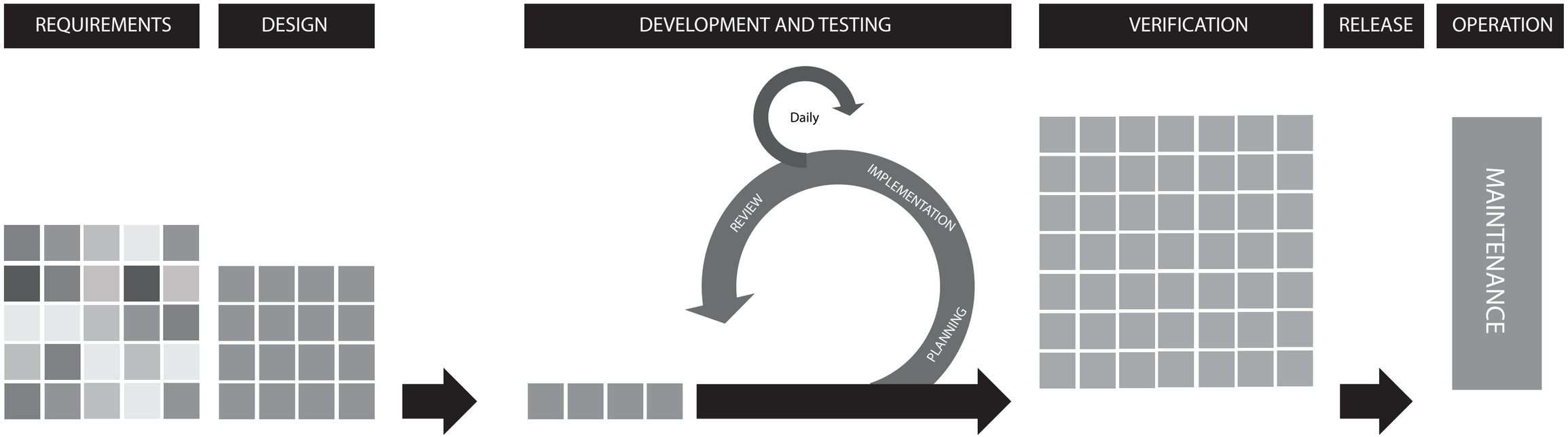
29 BSIMM. Building Security In Maturity Model. 2022. Available from: https://www.bsimm.com/

Appendix 1. Security activities require implementation or improvements

Activity [26]	Activity descriptions [29]	Practice area [26]	Phase	Category [27]	Activity type [14]	Urgency
Identify open source components	Use pipeline integratable tools to discover open source components and their old versions or to find components duplicate versions. Some platforms providers have stared to offer visibility into component metadata (SBOM). [29]	Standards & Requirements, Level 2	Requirements, Verification	Prevention	Automation	Medium
Control open source risk	Open source components' risk management includes several efforts: responding identified vulnerablity, controlling licenses, restricting use of components. [29]	Standards & Requirements, Level 3	Requirements, Verification, Operation	Prevention	Automation	Low
Identify potential attackers	Understanding attackers motivation and technical capabilities is important. Attackers' profiles definition is carried out periodically, and they are used in design reviews. [29]	Attack Models, Level 1	Design	Prevention	Manual	Low
Use attack intelligence	Organizations gather knowledge about the attacks and vulnerabilities, and share information for learning purposes. [29]	Attack Models, Level 1	Design	Prevention	Manual	Low
Build attack patterns and abuse cases tied to potential attackers	Creation of attack patterns and abuse cases prepares the organization for possible attacks and enables the protective measures development. Attack stories are also used in penetration testing and design reviews. [29]	Attack Models, Level 2	Design	Prevention	Manual	Low
Create technology-specific attack patterns	Creation attack patterns are relevant for organization technologies. Organizations can collect to the catalog used cryptographic methods and their exploitation. [29]	Attack Models, Level 2	Design	Prevention	Manual	Low
Use a top N possible attacks list	Creation of attack type lists for using to carry out the changes. Top N-list is updated rarely but regularly. [29]	Attack Models, Level 2	Design	Prevention	Manual	Low
Collect and publish attack stories	Collecting success and failure attack stories for training and discussing purposes. [29]	Attack Models, Level 2	Design	Prevention	Manual	Low
Use automation to imitate attackers	Mimicking attackers’ actions needs customized tools to consider infrastructures, technologies, and configurations in use. DevOps development method allows tools' embedding into pipelines and automation. [29]	Attack Models, Level 3	Design	Prevention	Automation	Medium
Monitor automated asset creation	Continuous control of network, software and infrastructure components cretation. Ensure proper automated monitoring activities. [29]	Attack Models, Level 3	Design	Prevention	Automation	High
Make code review mandatory	Code review can be set as mandatory, so that for instance unacceptable review results stop automatically release process. [29]	Code Review, Level 1	Implementation	Detection	Automation	Medium
Use automated tools with tailored rules	By using custom rules in automated testing tools it is easy to verify coding standards compliance or find middleware components' security flaws. [29]	Code Review, Level 2	Implementation	Detection	Automation	Medium
Enforce coding standards	Secure coding standards violation can cause code rejection and move code back to the development for fixing. Rejection can cause denied pull request, breaked builds, failed in quality check or code directed to other workflow. Programming lanquage specific coding standards are monitored with tailored rules of SAST tools. [29]	Code Review, Level 3	Implementation	Detection	Automation	Medium
Ensure QA performs boundary value condition testing	Tests using unexpected inputs such as boundary values and exploring application operation to find weaknesses. [29]	Security Testing, Level 1	Verification	Detection	Automation	High

Drive tests with security requirements and security features	Security features can be tested in the same way that other functionalities. Administrative accounts' security mechanisms for example lockout, transaction limits, entitlements must be tested with expected and unexpected inputs. [29]	Security Testing, Level 1	Verification	Detection	Automation	Medium
Integrate opaque-box security tools into the process	Opaque-box testing tools (e.g. web application scanners) can be integrated into the development process with internal cloud-based toolchains. [29]	Security Testing, Level 1	Verification	Detection	Automation	Medium
Share security results with team	Share security testing such as threat modelling, code reviews, penetration testing and composition analysis results with all relevant teams. [29]	Security Testing, Level 2	Verification	Detection	Manual	Medium
Include security tests in automation	Security tests should be automated and running beside performance, functional and other tests. Automated tests can be provoked manually or automatically as a part of pipeline processes, outside pipeline performed manual tests not providing feedback on time. [29]	Security Testing, Level 2	Verification	Detection	Automation	Medium
Perform fuzz testing customized to application APIs	Fuzzing tests include built-in intelligence from application business logic and API interfaces. [29]	Security Testing, Level 2	Verification	Detection	Automation	High
Drive tests with risk analysis results	Use risk analysis results to lead QA test creation. Created risk profiles can be used in definition and implementation of adversarial tests. [29]	Security Testing, Level 3	Verification	Detection	Manual	High
Leverage coverage analysis	Implement a code coverage analysis which increases depth of security testing. It is important to know how widely test cases use code. [29]	Security Testing, Level 3	Verification	Detection	Manual	High
Begin to build and apply adversarial security tests	Testers moving to use attacker's point of view to try to break the system with abuse cases-based test cases. Security policies, attack intelligence, standards, and N attack lists create a basis for abuse and misuse cases creation. [29]	Security Testing, Level 3	Verification	Detection	Manual	Medium
Implement event-driven security testing in automation	Event-driven testing is automation implemented to the pipeline that is triggered when new code is put in a repository. Automation activates testing after defined conditions are reached. [29]	Security Testing, Level 3	Verification	Detection	Automation	High
Use penetration testing tools internally	Implement internal penetration testing tools to an available development process. Automated tools complete manual testing and improve testing process efficiency. There are products for application and network penetration testing. [29]	Penetration Testing, Level 1	Verification, Release	Detection	Manual	High
Feed results to the defect management system	The DevOps team can follow other testing activity results in real-time in social tools such as JIRA and Slack. Security-related findings and change requests should be directed to the development team more quickly, although requests are also a part of the vulnerability management process. This process improves security and highlights importance of security vulnerabilities fixing. [29]	Penetration Testing, Level 1	Verification	Detection	Automation	High
Identify software defects found in operations monitoring and feed them back to development	Service operational monitoring can identify defects in the production environment, and all found defects must be fed to the development team backlog. Production environment defects enable learning and change behaviour in the organization. [29]	Configuration & Vulnerability Management (CMVM), Level 1	Operation	Response	Automation	Medium
Prevent software bugs found in operations	Preventing reoccurence of security defects is avoided with a systematic incident management process where postmortem feedback reports are written for the SSDL team. Defects proper root-cause analysis exposes at which stage of the application life cycle an error has slipped into the production environment. Operational experiences is lead SSDL process changes. [29]	Configuration & Vulnerability Management (CMVM), Level 3	Operation	Prevention	Manual	Low
Automate operational infrastructure security verification	Service infrastructure automated security verification operations are mandatory. [29]	Configuration & Vulnerability Management (CMVM), Level 3	Operation	Response	Automation	Medium

Appendix 2: Application development and operation process security activities requiring improvements



REQUIREMENTS	DESIGN	DEVELOPMENT	VERIFICATION	RELEASE	OPERATION
1. Identify open source components (A) 2. Ensure proper communication	1. Use automation to imitate attackers (A) 2. Monitor automated asset creation (A) 3. Ensure proper communication	1. Make code review mandatory (A) 2. Use automated code review tools with tailored rules (A) 3. Enforce coding standards (A) 4. Ensure proper communication	1. Automated security tests (A) 2. Opaque-box testing tools (A) 3. Use security features, event-driven, boundary values, adversarial and fuzz testing. (A) 4. Code coverage analysis 5. Share security test results 6. Use penetration testing tools internally 7. Feed test results to the defect management (A) 8. Identify open source components (A)	1. Use penetration testing tools internally	1. Identify defects and feed them back to development (A) 2. Automate operational infrastructure security verification (A) 3. Identify open source components (A) 4. Ensure proper communication