The present work was submitted to
the Research Group
Software Construction

of the Faculty of Mathematics,
Computer Science, and
Natural Sciences

Master Thesis

Security and Hardening in Continuous Delivery

presented by

Julian Schwarz

Aachen, September 30, 2019

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Dipl.-Inform. Andreas Steffens

Statutory Declaration in Lieu of an Oath

The present translation is for your convenience only.
Only the German version is legally binding.

I hereby declare in lieu of an oath that I have completed the present Master's thesis entitled

Security and Hardening in Continuous Delivery

independently and without illegitimate assistance from third parties. I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Official Notification

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whosoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 to 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

I have read and understood the above official notification.

Eidesstattliche Versicherung

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

Security and Hardening in Continuous Delivery

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, September 30, 2019 (Julian Schwarz)

Belehrung

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicher ung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen.

Aachen, September 30, 2019 (Julian Schwarz)

Acknowledgment

I would like to thank my supervisor Andreas Steffens for the continuous guidance and assistance during this thesis including weekend corrections and 24/7 email support.

Additionally, I would like to thank all people at our industrial partner who took their part in this thesis. Those include the two members of the DevOps team as well as the security expert who dedicated their time to the validation and evaluation of this thesis.

Julian Schwarz

Abstract

Continuous Delivery (CD) as well as the term DevOps have become increasingly more prevalent in recent years and play an important role in today's modern agile development methods. Unfortunately, security is often disregarded in an agile development process. Additionally, traditional security methods are not suited for the continuous releases. Thus, a problem we analyse in this thesis is the integration of security into CD by assessing the implementability of the security standard DIN ISO/IEC 27001 with CD. At that, we create a list of functional requirements delivery systems have to fulfil in order to comply to the ISO 27001. Additionally, we create models which serve as a guidance for the actual implementation, respectively realization. In the end, we use those requirements to perform an analysis of our industry partner's CD regarding the state of realization and to gather further insights into the ISO 27001.

Contents

List of Tables

List of Figures

1 Introduction

> Innovation is hard. It really is. Because most people don't get it. Remember, the automobile, the airplane, the telephone, these were all considered toys at their introduction because they had no constituency. They were too new.
>
> Nolan Bushnell

Contents

Continuous Delivery (CD) and the closely related term DevOps are becoming increasingly more prevalent and used. *Continuous Delivery (CD)* plays an important role in today's modern agile development methods by empowering these with shorter development cycles. DevOps on the other hand supports and implements CD with the right tooling and it does this with increasing responsibility taking e.g. the rise of *Infrastructure as Code (IaC)* into consideration which gained an important role for DevOps. Security is often not considered in this accelerated development cycle. Systems get more complex every day and security is usually not embedded into the development process. It is often put in second place and is only considered with some more attention on release of a certain piece of software. The problem is that CD features continuous releases for which traditional security methods are not suited. Thus we need to integrate security into the CD cycle. In recent years the new terms DevOpsSec or SecDevOps were formed which address exactly this problem[MCP17].

On the other hand, there exists a variety of security standards for information systems like the ISO 27001, the BSI200-1 which is regularly adjusted to the ISO 27001, etc. for whom companies can get certified. However, often not considered is the realization of those standards as a responsibility of the DevOps team, respectively there is not much literature regarding DevOpsSec in combination with security standards[MO16].

In this thesis we focus on the ISO 27001 as an information security standard for which we assess its implementability with CD, respectively want to answer the question which controls of the ISO 27001 can be supported (only) with CD. Thereby, we also assess the realizability to the best of our knowledge and give suggestions on how to implement the extracted requirements. The goal is to create a model for assessing and reviewing the current situation inside an organization regarding their current state of CD and how much of the ISO 27001 they already implement. Moreover, it serves as a guide to increase

the security through improvement of the CD process.

In the end, we perform an evaluation which includes a validation of our assessment of the implementability of the controls of the ISO 27001 to CD. Furthermore, we analyse, respectively discuss the current degree of ISO 27001 compliance of the CD as it is used by our industry partner.

At that, we deliberately focus on CD instead of DevOps, since it gives us a clearer, more abstract (CD as a process) but in the end also a better definable scope.

This work is done in cooperation with an industry partner which maintains multiple locations in Germany and across the globe. It offers various services in software, hardware and engineering. More precisely, this thesis is written for the Operations & Engineering team being part of the ISO 27001 certification of the company which is also the reasoning of our choice of the ISO 27001.

1.1 Structure of this Thesis

All necessary information and context for this thesis is given in chapter 2. This includes definitions for CD, DevOps and introduces several security standards. In Chapter 4 we then present security related work for DevOps which includes terminology like DevOpsSec and continuous security. Eventually, in chapter 3, we list the challenges followed by related work in chapter 4. In chapter 5 we then formulate our research questions based on the problem statement and the existing related work. As the central part of this thesis, in chapter 6 we give a detailed analysis of the ISO 27001 in in order to answer the research questions. Chapter 6 also comprises the different models based on the detailed analysis of the ISO 27001 and a list of realization suggestions for every functional requirement. Finally, chapter 7 deals with a discussion of the *Statement of Applicability (SoA)* involving 2 members of the DevOps team of our industry partner. Additionally, we discuss our models there. Chapter 8 then summarizes the most important aspects of this thesis. Furthermore, it deals with the threats to validity of this work and future work.

2 Background

Excellence is a continuous process and not an accident.

A. P. J. Abdul Kalam

Contents

In this chapter, we cover the basic definitions and frameworks needed to built upon in the further course of this thesis. There exists a huge number of security frameworks, however, we can not cover all of them in this section. Therefore, we focus on the most important and well-known ones such as the BSI-Standard 200-1 (see section 2.6.2), but also the EU *General Data Protection Regulation (GDPR)* (see section 2.6.3) and naturally the ISO 27001 (see section 2.6.1) which is an integral component of this thesis. Moreover, we shortly deal with the *Cloud Security Alliance (CSA)* (see section 2.7) which consolidates many security standards, including BSI 200-1 and the ISO 27001-27005.

2.1 Information Security Management System

According to the BSI-Standard 200-1[Bsia] a management system

> embraces all the policies pertaining to supervision and management for the purpose of achieving the organisation's objectives.

An *Information Security Management System (ISMS)* thereby is part of the management system and specifies

> the instruments and methods that the management level should use to clearly manage (plan, adopt, implement, supervise and improve) the tasks and activities aimed at achieving information security.

This includes the essential components *management principles*, *resources*, *employees* and the *security process*.

2.2 Continuous Delivery

CD is a software engineering approach which focuses on reducing cycle times of software deployment. According to Humble et al.[HF10], software releases should be a low-risk, frequent, cheap, rapid and predictable process. Furthermore, it ensures that software can be released at any time[Che15]. Humble et al. introduce the following release antipatterns which stay in contrast to CD.

The first antipattern is named *deploying software manually*. This antipattern is identified by frequent manual involvement in the build, respectively release process, e.g. if the release process requires frequent corrections or manual testing is required to confirm that the application is running. Steps in the pipeline are often considered to be separate and atomic, and different individuals perform different steps which leads to different ordering and timing of the steps. Resulting disadvantage of such a manual deployment comprise of among others the missing reproducibility and therewith missing reliability, the need to maintain deployment documentation and other waste of resources, since performing manual deployments is boring and repetitive, yet requires a significant degree of expertise. The remedy is to automate deployment[HF10].

The second antipattern *deploying to a production-like environment only after development is complete* is identified e.g. if before a release, testers tested the system only on development machines or if releasing into staging is the first time that operations people interact with a new release. Disadvantages as a result of this antipattern among others comprise the detection of new bugs when an application is deployed into staging, the documentation of the general process (steps) of deploying an application to staging often misses steps since it was never performed and furthermore the steps itself often are erroneous, because they were never performed. Here, the remedy is to integrate the testing, deployment, and release activities into the development process[HF10].

The last antipattern is called *manual configuration management of production environments*. It is identified by failing deployments to production even though multiple deployments to staging were successful. Furthermore, it is identified by a long preparation time of the operations team for an environment for a release, or by servers in clusters having different versions of operating systems, etc.. Finally, the remedy of this antipattern is to apply all aspects of each testing, staging and production environments, the configuration of any third-party elements of a system from version control to an automated process.

In the end a reduced cycle time also leads to less opportunity costs associated with not delivering software and it allows to verify features and bugfixes with respect to their usefulness[HF10].

Humble et al. state that in order to achieve a low cycle time and high quality, automated and frequent releases of software are required. Frequent, to have as small deltas between releases as possible in order to reduce the risk associated with releasing and roll-backs

of software. Feedback, especially fast one is extremely important in order to e.g. allow delivery teams to fix issues.

Finally, in order to classify and therefore allowing organizations to assess and improve their CD, the following section deals with two maturity models for CD in which levels, respectively classifications are defined. Moreover, the maturity models give a good overview of the general areas, but also lay out concrete aspects which can or should be realized for CD. This is helpful, since the definition and approach of a motivation for continuous delivery in the text above is quite general with the aim to introduce automated and frequent releases. However, it does not give any hint or even concrete aspects regarding their actual realization, because it is in the nature of CD being a complex problem with various kinds of areas to work on. This is where the maturity models can help.

2.2.1 Maturity Models

In the following we deal with 2 maturity models for continuous delivery.

Rehn et al.

Rehn et al. decided for their continuous delivery model shown in Figure 2.1 to introduce five levels of maturity and five categories. The levels range from base to expert whereas the categories comprise *Culture & Organization*, *Design & Architecture*, *Build & Deploy*, *Test & Verification*, and *Information & Reporting*. The model should serve as an evaluation of a company's maturity in order to identify actions which increase maturity most efficiently[BRP13]. To give an orientation, Rehn et al. state that the base level is where they see most organizations as of 2013. They consider organizations to benefit from the larger effects at the intermediate maturity level. After all, the levels are not mandatory stages which have to be passed in sequence, but rather serve as an orientation. Rehn et al. suggest to keep the maturity level fairly even over all categories and to favour incremental improvements over big changes in order to increase the acceptance in the organization.

Humble et al.

Humble et al. created a maturity model whose main focus is to reduce cycle times, to reduce defects, to increase the predictability of the software delivery lifecycle, to improve the ability to adopt and maintain an attitude of compliance to any regulatory regime, to improve the ability to determine and manage the risks associated with software delivery effectively, and to reduce costs. Therefore, they recommend to apply the Deming cycle which states to first plan, then do, then check and finally to act, respectively to review[HF10]. Furthermore, they carefully address all the roles involved in the delivery process including their interactions. The complete maturity model is shown in Table 2.1

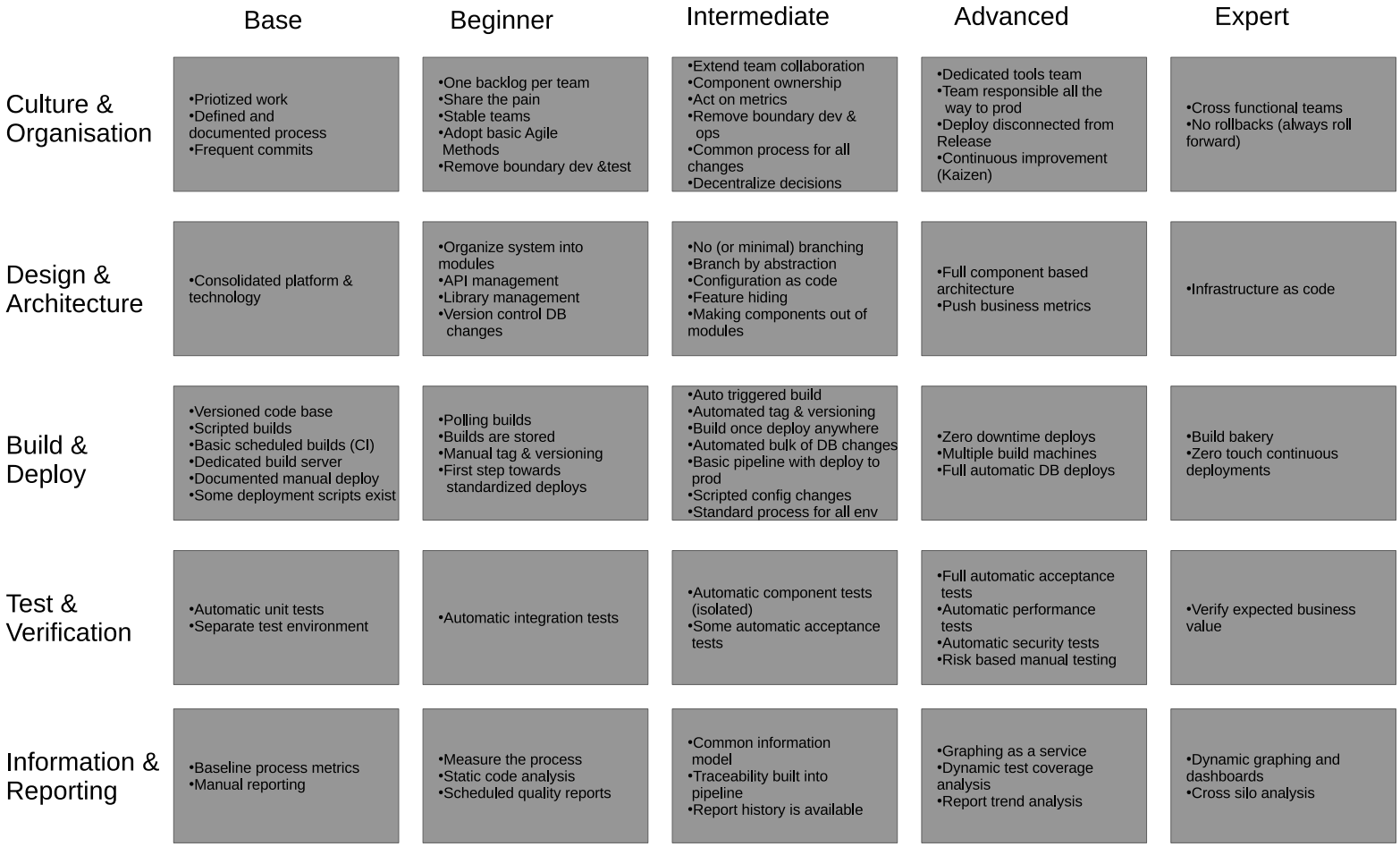

Figure 2.1: Maturity model for CD (adapted from [BRP13])

Practices	**Build management and continuous integration**	**Environments and deployment**	**Release management and compliance**	**Testing**	**Data management**	**Configuration management**
Level 3 - Optimizing: Focus on process improvement	Teams regularly meet to discuss integration problems and resolve them with automation, faster feedback, and better visibility.	All environments managed effectively. Provisioning fully automated. Virtualization used if applicable.	Operations and delivery teams regularly collaborate to manage risks and reduce cycle time.	Production rollbacks rare. Defect found and fixed immediately.	Release to release feedback loop of database performance and deployment process.	Regular validation that CM policy supports effective collaboration, rapid development, and auditable change management process.
Level 2 - Quantitatively managed: Process measured and controlled	Build metrics gathered, made visible, and acted on. Builds are not left broken.	Orchestrated deployments managed. Release and rollback processes tested.	Environment and application health monitored and proactively managed. Cycle time monitored.	Quality metrics and trends tracked. Non functional requirements defined and measured.	Database upgrades and rollbacks tested with every deployment. Database performance monitored and optimized.	Developers check in to mainline at least once a day. Branching only used for releases.
Level 1 - Consistent: Automated processes applied across whole application lifecycle	Automated build and test cycle every time a change is committed. Dependencies managed. Re-use of scripts and tools.	Fully automated, self-service push-button process for deploying software. Same process to deploy to every environment.	Change management and approvals processes defined and enforced. Regulatory and compliance conditions met.	Automated unit and acceptance tests, the latter written with testers. Testing part of development process.	Database changes performed automatically as part of deployment process.	Libraries and dependencies managed. Version control usage policies determined by change management process.
Level 0 - Repeatable: Process documented and partly automated	Regular automated build and testing. Any build can be re-created from source control using automated process	Automated deployment to some environments. Creation of new environments is cheap. All configuration externalized / versioned.	Painful and infrequent, but reliable, releases. Limited traceability from requirements to release.	Automated tests written as part of story development.	Changes to databases done with automated scripts versioned with application.	Version control in use for everything required to recreate software: source code, configuration, build and deploy scripts, data migrations.
Level -1 - Regressive: Processes unrepeatable, poorly controlled, and reactive	Manual processes for building software. No management of artifacts and reports	Manual process for deploying software. Environment-specific binaries. Environment provisioned manually.	Infrequent and unreliable releases.	Manual testing after development.	Data migrations unversioned and performed manually.	Version control either not used, or check-ins happen infrequently.

Table 2.1: Maturity model for CD (adapted from [HF10])

Figure 2.2: Example of a deployment pipeline (adapted from [HF10])

2.3 DevOps

Bass et al. provide the following definition for DevOps[BWZ15]:

> DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.

The relation between DevOps and CD is that DevOps uses CD in order to reduce the time between commit and production change.

2.4 Deployment Pipeline

There are different definitions of the pattern *deployment pipeline*. Humble et al.[HF10]. define a *deployment pipeline* as an

> automated implementation of your application's build, deploy, test, and release process.
>
> The purposes of a *deployment pipeline* are:
>
> - visibility of process
> - improvement of feedback, respectively having feedback as early as possible
> - deploying and releasing any version to any environment fully automated
>
> An example for a *deployment pipeline* according to the used terminology by Humble et al.[HF10] is shown in Figure 2.2.
>
> However, there exist further different definitions in literature. Thus, in this thesis, we are going to use the consistent terminology defined by Steffens et. al.[SLD18]. It consists out of three terms which specify the term *deployment pipeline*.
>
> - *Software Delivery Model*: A deployment pipeline models a delivery process
> - *Software Delivery System*: A deployment pipeline is an integrated software system
> - *Software Delivery Process*: a deployment pipeline exhibits characteristics of a process

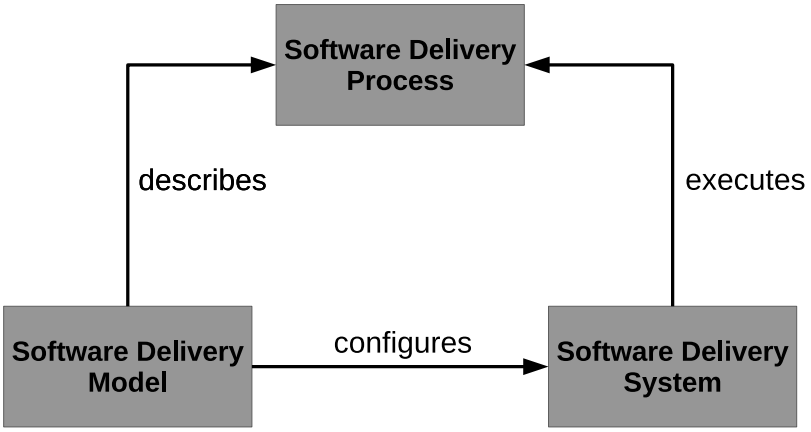

Figure 2.3: Relations between Software Delivery Process, Model and System (adapted from [SLD18])

Figure 2.3 illustrates the relationships between these terms.

2.5 Security Framework Design Principles

There are many reoccurring principles and corresponding controls in security frameworks and standards in general and the frameworks covered in this thesis pose no exception of that.

In his book *Hands-On Security in DevOps*[Hsu18] Tony Hsu deals among others with security architecture and design principles which we address in the following to give a profound understanding of security frameworks.

He introduces two key concepts in security architecture design principles: *security by design* and *privacy by design*. They differ in the former having unauthorized access to the system as a primary concern whereas the latter deals with the authorized processing of private data. For both, he lists a number of principles and a number of example controls which are shown in Table 2.2 and can be found in the various security frameworks we cover including the ISO 27001 (specifically in this thesis in section 6.1).

2.6 Well-known security frameworks

Even though this thesis is only based on the the ISO 27001, in the following we also give an overview over some more well-known security frameworks, respectively legislations.

Concept	Security by design	Privacy by design
Principles	• Minimize attack surface area • Establish secure defaults • Principles of least privilege • Principle of defense in depth • Fail securely • Don't trust services • Separation of duties • Avoid security by obscurity • Keep security simple • Fix security issues correctly	• Collection Limitation Principle • Data Quality Principle • Purpose Specification Principle • Use Limitation Principle • Security Safeguards Principle • Openness Principle • Individual Participation Principle • Accountability Principle
Examples of controls	• Access control • Unsuccessful login attempts • Session control • Timestamps • Non-repudiation • Configuration change control • Audit security events • Cryptographic module • Incident monitoring • Error handling	• Cookie • Anonymity • Consent • Obfuscation • Restrict • Notify and inform • Authentication • Minimization • Separation • Encryption • Data masking

Table 2.2: Key concepts in security architecture design principles (adapted from [Hsu18])

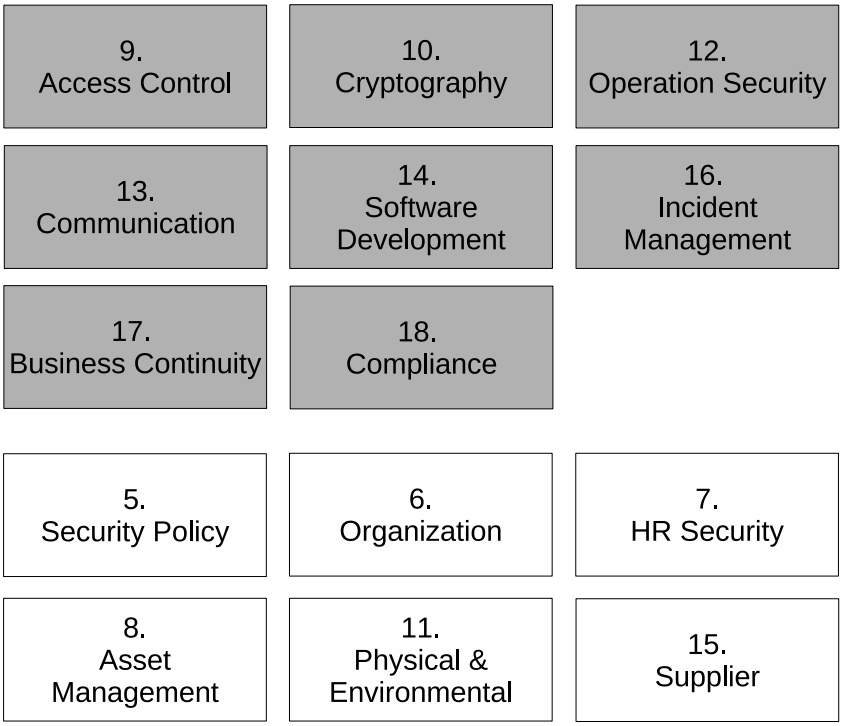

Figure 2.4: Areas of the ISO 27001 norm (adapted from [Hsu18])

2.6.1 ISO 27001

The DIN ISO/IEC 27001, here ISO 27001, is an international standard whose scope comprises controls for establishing, implementing, maintaining and continually improving an information security system[Isob]. It is designed to be used for internal and external assessment of an organization's ability to meet its own information security requirements. It is important to note that the ISO 27001 is an ISMS which provides a complete set of security management programs. Thus, it does not specify a technical security approach[Hsu18].

In Figure 2.4 an overview of the areas of the ISO 27001 is shown, as provided by Tony Hsu[Hsu18].

Figure 2.4 shows that the ISO 27001 norm can be split into two areas of responsibility. The upper area which comprises of topics which are more related to the responsibilities of the DevOps team and the lower area which is more related to company, respectively organization security policies[Hsu18].

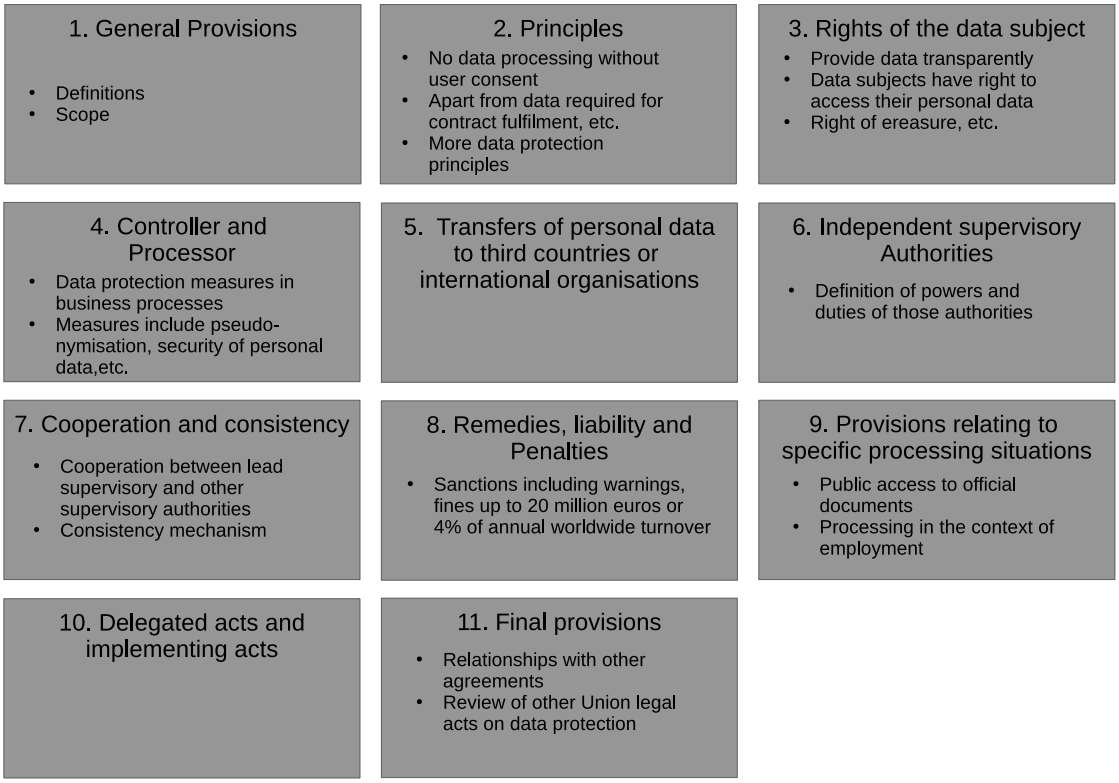

Figure 2.5: Areas of the GDPR[Gdp]

2.6.2 BSI 200-1

The BSI 200-1 standard is a security standard which is intended to form a reasonable basis for persons in charge of security to ensure that the aspects of information security management are adequately taken into account in their organization and projects[Bsia]. Thereby, it provides information in a step-by-step fashion. Similar to the ISO 27001 it describes how an ISMS can be designed in terms of general methods and regarding the initiation and management of information security in an organization.

2.6.3 EU GDPR

Not quite in the area of security but more in the area of privacy lays the *European Union (EU)* GDPR[Gdp]. It contains regulations on data protection and privacy in EU law. One of the main goals is to give individuals control over their own personal data. The GDPR comprises of eleven chapters as shown in Figure 2.5.

In contrast to the other two security frameworks, the EU GDPR has to be implemented, respectively followed by all companies in the European economic area since it is a regulation in EU law

2.7 Cloud Security Alliance

The CSA is a non-profit organization which promotes the use of best practices for security within cloud computing. They consolidated most security compliance methods in the *Cloud Controls Matrix (CCM)*[All] including the ISO. This has the advantage that by referring to the CCM, we can assure that all other security compliance standards are also met. To give an idea of how the CCM looks, we created a more abstract representation shown in Table 2.3.

Control Domain	**Control Specification**	**Architectural Relevance**	**Corp Gov Relevance**	**Cloud Service Delivery Model Applicability**	**Supplier Relationship**	**Scope Applicability**
Application & Interface Security - Application Security	Applications and interfaces shall be designed, developed, and deployed in accordance with industry acceptable standards and adhere to applicable legal, statutory, or regulatory compliance obligations	Network, Compute, Storage, App, Data	None	SaaS, PaaS, IaaS	Service Provider	NIST SP 800-53 R3 SC-5, NIST SP 800-53 R3 SC-6, NIST SP 800-53 R3 SC-7, NIST SP 800-53 R3 SC-12, NIST SP 800-53 R3 SC-13, NIST SP 800-53 R3 SC-14, A.11.5.6, A.11.6.1, A.12.2.1, A.12.2.2, A.12.2.3, A.12.2.4, A.12.5.2, A.12.5.4, A.12.5.5, A.12.6.1, A.15.2.1
Application & Interface Security - Customer Access Requirements	...	...	...	...	...	

Table 2.3: Exemplary excerpt of CCM[All]

3 Problem Statement

> I don't want to expose the intricacies of my work so people can understand how I did it.
>
> Billy Crudup

Contents

3.1 Challenges

The goal of this thesis is to provide guidance and models in order to help an organization's CD to maximally support and comply to the ISO 27001 and to increase security with CD in general.

Therefore, we have to identify the controls of the ISO 27001 which lay in the responsibility of the DevOps team or where the DevOps team can, respectively has to provide support for. To identify the requirements, we have to fit something as diverse as the ISO 27001 into a well-defined model, respectively requirement specification for CD.

3.1.1 Intricacy of Controls and Scope

The first challenge is the intricacy which every control of the ISO 27001 entails. Many controls are applicable to artefacts and systems or to systems and the CD process itself or even to all of that.

Taking as an example a requirement which is applicable to the CD system. Consequently this requirement also has to be fulfilled by a CD system which is built in a CD system, thus every CD system also has to support the verification of this requirement. This obviously is an infinite loop. So, we also

face the challenge to define clear, consistent, useful and sensible boundaries regarding the applicability of requirements to CD.

3.1.2 Fuzziness of Controls

Closely related to the intricacy of the controls is this second challenge namely the fuzziness of the controls which makes it not only difficult to assess the applicability with respect to our scope, but it makes it also difficult to decide which aspects we have to further specify on formulating a functional requirement and which aspects we let specify our functional requirements.

3.1.3 CD as ISMS

The third challenge is the fact that the ISO 27001 is not designed to serve as a requirements specification for anything other than a management system. It explicitly does not specify a technical security approach, however, we apply the requirements to a technical approach namely CD with the reasoning that we consider CD to be used as an ISMS.

3.1.4 Helpful Models

Eventually, due to our goal to give organizations not only a bunch of requirements, but also a guidance on how to approach the extracted requirements the last and fourth challenge comprise the creation of models to be used in practice. Furthermore, we also want to evaluate security in the context of CD in general.

4 Related Work

> You are an essential ingredient in our ongoing effort to reduce Security Risk.
>
> Kirsten Manthorne

Contents

The overall goal of this thesis is to automate security, respectively more specifically to implement as much of the ISO 27001 with the help of CD. Hence, in the following we deal with topics like DevOpsSec, security frameworks building upon the ones presented in chapter 2 and continuous security. Furthermore, we deal with the *Open Web Application Security Project (OWASP) Software Assurance Maturity Model (SAMM)* as a reference of a security maturity model.

4.1 DevOpsSec

DevOpsSec is a term which arose in recent years. It exists in different permutations of *dev*, *sec* and *ops*, but in this thesis we will stick to *DevOpsSec*. We introduced DevOps in chapter 2 as a practice which uses CD to reduce the time between a commit and the deployment to production. Conservatively, security was paid attention to at the point right before the releasing of software. With the reduced cycle times, security has to be integrated in the DevOps process[MCP17].

Myrbakken et al. performed a literature review[MCP17] around the term DevOpsSec. They suggest that there is a consensus regarding the definition

of DevOpsSec which is seen as[MCP17]:

> a necessary expansion to DevOps, where the purpose is to integrate security controls and processes into the DevOps software development cycle and that it is done by promoting the collaboration between security teams, development teams and operations teams.

They also identified practices that characterize DevOps. Those comprise continuous testing, monitoring and logging, security as code, and red-team and security drills.

In the following, we give an overview of the view on DevOpsSec of the book *Hands-On Security in DevOps: Ensure continuous security, deployment and delivery with DevSecOps* by Tony Hsu[Hsu18]. We specifically consider it important related work since it not only deals with security in DevOps, but also deals with the ISO 27001 in the context of DevOps on which we focus specifically.

4.1.1 Hands-On Security in DevOps

Tony Hsu introduces several driving forces and challenges for DevOpsSec. Those comprise security compliance methods such as the ISO 27001, which involves responsibilities for the DevOps team as shown in Figure 2.4, but also other security compliance methods like the ones consolidated by the CSA (see section 2.7). Further drivers and challenges are new technologies like Docker which introduce new security risks, but also IaC in general which speeds up application deployment but shifts the responsibility of system security to the DevOps team[Hsu18]. Finally, speaking of IaC, we already presented continuous delivery maturity models in section 2.2.1 (for which IaC is e.g. required at the latest for the intermediate level in the model by Rehn et al.). Correspondingly, Tony Hsu presents a less comprehensive but more concrete model for maturity of DevOps practices consisting of the three levels *Continuous Integration*, *Continuous Delivery* and *Continuous Deployment*. IaC is required for the continuous delivery level. Higher levels of maturity of DevOps practices thereby offer the possibility to integrate security in the build process without changing existing development and to cope with the challenge of short release cycles which might not allow for a full cycle of security testing.

Having discussed the motivation for DevOpsSec, Tony Hsu continues with security goals and metrics. We deal with them here to give an overview of what the responsibilities of the DevOps team can encompass. He states that the end goal of security for any organization is to secure customer digital assets. Aligned with that goal have to be the security goals of all teams in an organization. In the following, we deal with the development team security

goals and the operational team security goals, because of their relation to DevOps.

Development Goals

The security goal of a development team is to deliver secure design and implementation.

Based on OWASP SAMM practices, the key aspects to consider during the development phase comprise *threat assessment*, *security requirements* and *security architecture and coding*. Additionally, those are reflected by the deliverables a development team can provide with a self-assessment, namely a threat modeling analysis report, a secure coding analysis report and a secure architecture.

Regarding threat assessment, Tony Hsu presents two threat modeling tools, respectively guidelines which are suggested for the project team, one captioned with *Knowledge-base of threats and mitigation* and one *Tools or threat modeling templates*. The former one can help the team to decide regarding the most relevant threats to the project from the knowledge list instead of starting from zero. As a knowledge base e.g. *Common Attack Pattern Enumeration and Classification (CAPEC)* can be used. The latter one enables the team to deliver consistent quality for threat modeling reports.

Operational goals

As with section 4.1.1, the SAMM allows for a categorization of the key operational goals. They comprise *issue management*, *environmental hardening* and *operational enablement*.

Issue management defines how security incidents are handled which is important for both the DevOps and the Dev team, since they should have a vulnerability process. *National Institute of Standards and Technology (NIST)* SP800-61 might serve as a reference here and introduces different stages for an incident handling action checklist. Typical security activities during security incident handling comprise *vulnerability received*, *internal/external communication*, *root/cause analysis*, *mitigation*, *deployment and verification*. All of those activities involve the DevOps team, e.g. by checking for well-known *Common Vulnerabilities and Exposuress (CVEs)* or introducing new firewall security policies.

Next, environmental hardening should be covered in an organization's security policy considering a secure configuration baseline and a constant monitoring mechanism including definitions on what should be scanned with which tools. Table 4.1 shows a small overview regarding monitoring.

Areas	Purpose	Open source tools
CVE	To understand if there are any publicly known vulnerabilities in the cloud services. Refer to `https://cve.mitre.org/`	OpenVAS, NMAP
Integrity monitoring	It determines if major system configuration files have been tampered with.	OSSEC
Secure configuration compliance	Secure configuration to meet industry best practices.	OpenSCAP
Sensitive information exposure	To review whether there is any personally identifiable information, keys, or secret leakage in the configuration files.	No specific open source tool in this area. However, we may define specific regular expression patterns to scan the sensitive info.

Table 4.1: Overview of constant monitoring mechanisms (adapted from [Hsu18])

Finally, operational enablement bridges the development team and the DevOps team. Typical activities e.g. involve packaging deployment to production, ensuring the integrity of every software release or securing configuration.

4.2 Practice of Security Standards

In section 2.6 we dealt with well-known security standards. In this chapter, we build upon them by dealing first with the *ISO 27002*, the code of practice which belongs to the ISO 27001 and second with the *BSI 200-2* which provides step-by-step guides for the realization of the *BSI 200-1* among others considering the size of the company.

4.2.1 ISO 27002

Based on the ISO 27001, the *ISO 27002* provides a reference for organizations regarding selection of controls within the process of implementing an ISMS. It is intended to be used in developing an individualized industry- and organization-specific information security guidelines. Furthermore, its scope comprises organizations which intend to implement commonly accepted information security controls[Isoa].

The ISO 27002 consists of security control categories. Each of those categories contain a control objective stating what is to be achieved and one or more controls that can be applied to achieve the control objective. The descriptions of the controls consists of the control itself, an implementation guidance and other information. Thereby, the controls match the ones from the ISO 27001. Besides that, the implementation guidance provides more detailed information to support the implementation of the control and to meet the control objective. However, the guidance may not be entirely suitable or even sufficient in all situations and may not fulfil the organization's specific control requirements[Isoa]. Finally, the *other information* paragraph provides further information that may be taken into consideration, e.g. legal consideration. Still, it is important to note that (as the ISO 27001) the ISO 27002 is also not a technical security approach.

4.2.2 BSI 200-2

Based on the BSI 200-1 (see section 2.6.2), the IT-Grundschutz BSI 200-2 standard is a comprehensive framework for an ISMS and provides step-by-step guides to develop information security management in practice. Those can be adapted to the requirements of organizations of various types and sizes which is implemented via the three methodologies *Standard Protection*, *Basic Protection* and *Core Protection*[Bsib].

Basic Protection in terms of the BSI 200-2 exists to get a broad and basic initial safeguard of all relevant business processes. It allows the implementation of the most important security requirements and can be enhanced by protecting all areas using Standard Protection or by protecting critical business processes using Core Protection.

After all, the implementation of the IT-Grundschutz can be certified according to ISO/IEC 27001, respectively is compatible to the ISO 27001 which means it also includes recommendations from e.g. the ISO 27002.

4.3 Security Development Lifecycle

The *Security Development Lifecycle (SDL)* is designed by Microsoft to help developers build secure software. It is born out of the need of customers to receive, respectively acquire secure software of their vendors. However, it is not limited to that, since security also influences reliability, taking e.g. a security mitigation that protects against denial of service attacks. This mitigation is also a reliability feature[HL06]. The SDL is proven to increase security and allowed Microsoft to become or be at least perceived as a security trendsetter in 2006. Its goals comprise the reduction of the number of security

vulnerabilities and privacy problems and the reduction of the severity of the remaining vulnerabilities.

Now, the SDL consists out of thirteen stages whom we do not deal with in detail. Though, to give an overview those comprise *Education and Awareness*, *Project Inception*, *Define and Follow Design Best Practices*, *Product Risk Assessment*, *Risk Analysis*, *Creating Security Documents, Tools, and Best Practices for Customers*, *Secure Coding Policies*, *Secure Testing Policies*, *The Security Push*, *The Final Security Review*, *Security Response Planning*, *Product Release* and *Security Response Execution*.

4.4 OWASP SAMM

The OWASP SAMM is an open framework for organizations to implement a software security strategy[Sam]. As such it shall always remain vendor-neutral and freely available for all to use. It categorizes security practices into four key business functions as shown in Table 4.2, in contrast to the Microsoft SDL which defines security practices during the development process.

Furthermore, it is defined with flexibility in mind which means that it can be used by organization of any size using any style of development. The SAMM is built on the following principles:

- An organization's behaviour changes slowly over time
- There is no single recipe that works for all organizations
- Guidance related to security activities must be prescriptive

Software security programs should adapt to this first aspect by being specified in small iterations. Moreover, it should be flexible to allow organizations to tailor their choices regarding the second aspect. Finally, all steps in building and assessing an assurance program should be simple, well-defined and measurable[Sam].

The building blocks of the SAMM are three maturity levels for each of the twelve security practices (seen in Table 4.2).

4.5 Continuous Software Engineering and Agile Methods

First in [FS14] Fitzgerald et al. state that in the last two decades, there has been a widespread recognition that increasing the frequency of certain critical activities helps to overcome many challenges. This can bee seen with practices like *release early, release often*[FS14]. Furthermore, continuous integration and the recent emphasis on DevOps is given as an example that to eliminate

Business Functions	**Security Practices**
Governance	• Strategy & Metrics • Policy & Compliance • Education & Guidance
Construction	• Threat Assessment • Security Requirements • Secure Architecture
Verification	• Design Review • Code Review • Security Testing
Deployment	• Vulnerability Management • Environment Hardening • Operational Enablement

Table 4.2: SAMM Overview with business functions belonging to software development (adapted from [Sam])

discontinuities between development and deployment the integration has to be continuous.

Apart from continuous activities which are important for software development in today's context like continuous planning, continuous integration, continuous delivery, continuous verification and testing, continuous use they also identified continuous security.

Fitzgerald et al. consider the entire software life-cycle and identified three main sub-phases in which they categorize their activities. *Business Strategy & Planning*, *Development* and *Operations*.

Continuous security is considered to be situated in the development phase together with e.g. continuous compliance. The need for continuous security arises among others from the need for security in projects applying agile methods in safety critical systems. Originally, agile methods were seen as being suited only for small projects in non-safety critical contexts. This has changed in recent times.

Merkow et al.[MR11] defined building blocks for continuous application software security. Those comprise *Training and Awareness*, *Reusable Security APIs*, *Security Frameworks*, *Software Security Tools*, *Security of COTS Software*, *Software Security Incident Management* and *Continuous Security Testing*, *Software Security Group*, *Non-functional Requirements*. In the following we would like to shortly go into detail on the last aspect which has yet to be covered.

Non-functional requirements are the quality, security and resiliency aspects of software[MR11]. Merkow et al. suggest to address the issue of non-functional requirements only showing up in requirements documents when they are deliberately added by establishing a requirements analysis process that treats non-functional requirements as equal citizens to functional requirements. Thereby, the formal Software Security Group should establish processes to assure that this regularly occurs in order to obtain well defined and reusable non-functional requirements. For that, agile development with its user stories is one method for requirements collection that is quickly applicable and could be used to capture both functional and non-functional requirements. As of Merkow et al., key non-functional requirements comprise availability, capacity, efficiency, extensibility, interoperability, manageability, maintainability, performance,portability, privacy, recoverability, reliability, scalability, security, and serviceability[MR11].

Another paper which focuses on security in agile development methods is the case-study *Scaling Agile Methods to Regulated Environments: An Industry Case Study*[Fit+13] written by Fitzgerald et al.. More specifically, it evaluates the suitability of agile methods in regulated environments. Examples for regulated environments are industries like automotive, aviation, food, medical devices, etc.. The result of that case-study is that agile is highly suitable when

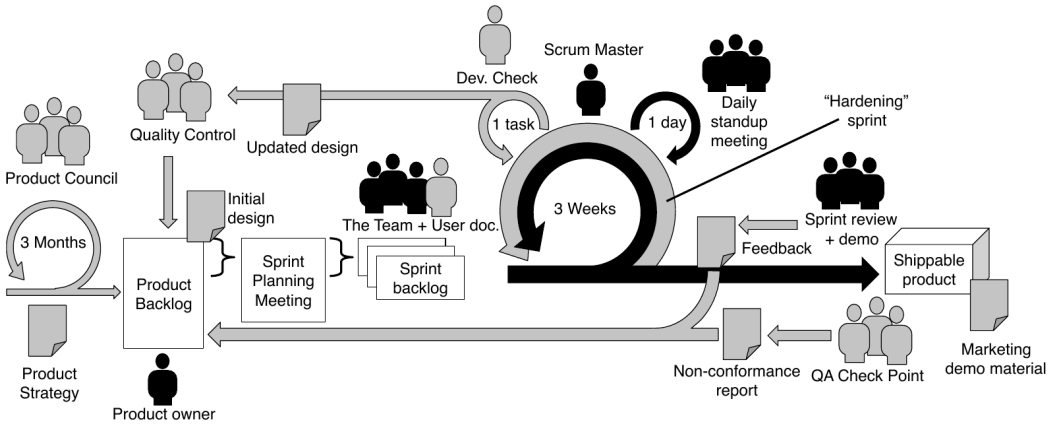

Figure 4.1: R-Scrum: Regulated Scrum implementation out of case study[Fit+13] (copied from [Fit+13])

tailored to meet the needs of regulated environments and supported with appropriate tools. The case-study company e.g. switched to an integrated toolset (Atlassian) which helped significantly to support full end-to-end traceability which usually is a significant overhead in regulated environments. Traceability thereby is defined as [Com+90]:

> The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match

Moreover, for verification and validation they also found a powerful toolset in continuous integration as quality enhancing. After all, they termed the new agile Scrum process for regulated environments *R-Scrum* which is shown in Figure 4.1.

4.6 Missing Requirements and Models

We dealt with DevOpsSec in general and Tony Hsu[Hsu18] even dealt with the ISO 27001 in this context. However, as seen in chapter 2, Tony Hsu only defines the areas of responsibility of the ISO 27001 sections which is not enough, since it still requires anyone wanting to implement the ISO 27001 to apply the controls to CD. Furthermore, Tony Hsu does not really exclude any sections, he just states the sections which lay rather in the responsibility of the DevOps team and those wo do not.

Regarding DevOpsSec there is nothing close to requirements. There is no literature extracting functional requirements out of the ISO 27001 for CD. Tony Hsu only describe goals in the context of DevOpsSec and give approaches on how to achieve those goals including technical means. However, this is neither comprehensive nor is it based on the ISO 27001. Furthermore, also the BSI 200-2, the ISO 27002 or the Microsoft SDL still do not formulate functional requirements in general, particularly none specific to CD. They rather define areas of responsibility, respectively building blocks for security in CD, but nothing fine-granular. The same applies to Merkow et al. even though they define building blocks for continuous application software security which is more in the direction of CD. However, it is also not based on the ISO 27001.

Regarding models, there do not exist any CD specific security-related CD maturity models or realization roadmaps. Even though the SAMM is, respectively includes a maturity model and one might even consider the SDL to be a model, both do not specifically consider CD.

5 Methodology

> Science is fun. Science is curiosity. We all have natural curiosity. Science is a process of investigating. It's posing questions and coming up with a method. It's delving in.
>
> Sally Ride

Contents

In this chapter, we define the general scope of this work based on chapter 1, chapter 3 and chapter 4.

We formulate the research questions regarding the problems which will guide through this thesis and define the artefacts we create in order to answer them.

5.1 Scope of Work

The scope of this work is to deduct functional requirements for the *delivery system* based on the controls listed in the ISO 27001. We chose the ISO 27001 because it is the certification our industry partner wants to comply with at the time of writing. As *delivery system*, we understand all systems which are involved in delivering software, however, do not have to be limited to that. Those comprise systems which we use by means of tools such as the version control system, the artefact storage, and the pipeline manager, but also artefacts (systems) which are created or used during this delivery process like production applications, or test systems, etc..

Furthermore, we give a detailed definition of how we use the terms *model*, *system* and *process*, based on the definition in section 2.4, to further specify our understanding of continuous delivery. Moreover, we use those terms to classify the controls of the ISO 27001.

The *process* comprises everything which is pre-definable and can be implemented in a process. This also includes aspects which conceptually fit,

respectively can be realized with a continuous delivery process, but which are currently not realizable in terms of technical realizability.

The *model* comprises everything which has to be set by the user and can not be implemented in the process.

The *system* comprises everything which is required to execute the process.

In section 6.1, we distinguish between applicability, implementability and functional requirement, which is why we do not consider technical realizability for any of the terms *model*, *process* and *system*, since they belong to implementability.

5.2 Research Questions

In the following, we define our research questions in order to define and to limit the scope of this thesis, but also to give it some structure.

- **RQ1:** *Which aspects of the ISO 27001 are relevant for the Software Delivery Process, the Software Delivery Model and the Software Delivery System (see Figure 2.3)? What are the resulting functional requirements?*

  This research questions is the core of our work and incorporates all but the last challenge (see section 3.1). We first have to assess the applicability of every ISO 27001 control to CD. Then we can perform the assignment to the finer-granular terminology of the term *deployment pipeline*: *model*, *process* or *system*. Thereby, we basically assess the implementability, since we decide where the ISO 27001 control can conceptually be implemented. Finally, we can formulate functional requirements for the delivery systems. After all, we do this in the form of a SoA.

- **RQ2:** *What is the relation of aspects between model, process and system? Is this in accordance with the ISO 27001?* We analyse the number of aspects which we identified as being related to model, process, or system. It shows us the relations between those numbers and allows us to discuss whether they are reasonable which also gives us the possibility to discuss our assignment. Furthermore, it shows us to which degree the ISO 27001 can be automated which is especially interesting as CD aims for a high automation degree.

- **RQ3:** *Are there relations, respectively dependencies between the identified functional requirements? Are there relations, respectively dependencies of the identified requirements to known maturity models for CD which can be used to derive a security maturity model?*

  First, we create a realization roadmap. This means we set the functional requirements in relation to each other based on effort. Second, we

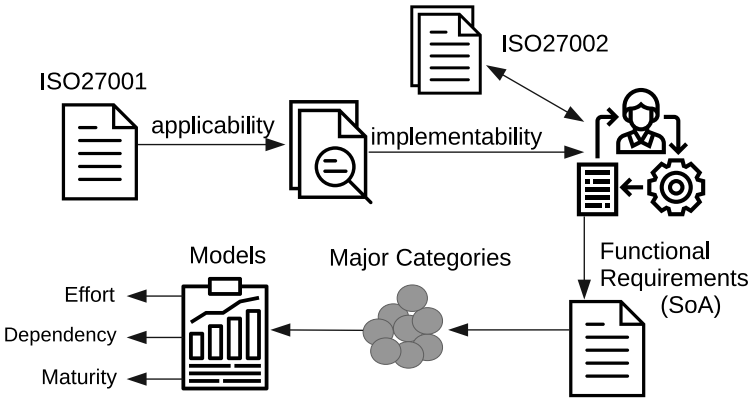

Figure 5.1: Overview of methodology

create a dependency model which shows the dependencies between the functional requirements. Third, based on the dependency model we then create a maturity model.

Finally, we create a mapping between our extracted ISO 27001 requirements for CD and the maturity levels of a CD maturity model in chapter 2. Eventually, we extend an existing maturity model for CD with our functional requirements.

The aim of this research question is to fulfil our goal to give organizations not only a list of functional requirements, but also guidance on how and when to realize them. With the last model, we want to identify the level of maturity of CD security requires to help organization which already implement CD evaluate which requirements they can already integrate in their CD process. Here, we face the fourth challenge (see section 3.1.4).

- **RQ4:** *Which of those aspects are realizable (at the moment)? How would one implement them (not only concept)? Can we prioritize them?*

  As a final research question, we assess each functional requirement which we got as a result of the SoA (see section 6.1) on realizability. Thereby, we create a small catalogue containing concrete realization suggestion. At that, we focus on delivery systems which are used at our industry partner. This research question shall give an idea of how companies can technically realize the stated functional requirements.

The complete process, respectively methodology from the ISO 27001 to the final models can be found in Figure 5.1. At that, the arrows forming a circle around the person illustrate that this is an iterative process and not a onetime transformation from the ISO 27001 to the SoA.

6 Results

> If the staff lacks policy guidance against which to test decisions, their decisions will be random.
>
> Donald Rumsfeld

Contents

6.1 Statement of Applicability

In the context of the *ISO* standards a well-known format regarding the applicability of controls is the SoA. Basically, the SoA lets an organization define for each control whether it is considered applicable in that organization or not. In the latter case, the organization has to give a well-reasoned explanation.

In the following, we use the format of the SoA to determine and document

three aspects. First, we also evaluate the applicability of each control with regard to whether it fits to the idea of CD. At that, we orient ourselves towards the criteria defined in section 6.1.1. Second, we evaluate the implementability of an applicable control. With implementability we mean whether it can be integrated in the process of CD which is implemented with the help of a delivery system. Therefore, we assign the respective control to the terms *model*, *process*, or *system*. There might be multiple assignments. Last, if the control is applicable and implementable, we define the functional requirement for the delivery system. Therefore, we mainly consider the external view, hence, we only consider requirements for the delivery system. When we speak of delivery system, we usually mean the build system (e.g. Jenkins), the version control system (e.g. Gitlab) and the artefact storage (e.g. Nexus). Only after that, we also deal with the requirements the artefacts and systems which are created with the delivery system have to fulfil. Naturally, for all aspects we assume that CD is perfectly implemented for all projects in order to be able to state which aspects can be implemented with CD in any way. Taken into consideration for the realization is among others the *ISO 27002* (see section 4.2.1).

6.1.1 Reasoning of Applicability of ISO 27001 to CD

In order to reason the applicability of the controls of the ISO 27001 to CD, we consider aspects applicable which:

- are supposed to be applied to any kind of software system or require systems to take part in a process / measure
- induce restrictions on any entity / functionality / other system which is typically involved in a CD system such as version control systems
- induce restrictions on data which is typically stored, processed or created in CD systems, like credentials, logs, source code, etc.
- are related to aspects which are already included by the three inclusion criteria above and not excluded by the two exclusion criteria

In contrast, we consider aspects not applicable which:

- are of a purely organizational nature, e.g. aspects which require guidelines, etc.. However, we include aspects which also induce the application/implementation of those guidelines
- match the first inclusion criteria above, but do not require a system to provide something actively, e.g. requiring documentation for all systems regarding certain aspects or user interactions with a system like *regularly review user access rights*, etc.

We might deviate from those general rules, respectively aspects. However, if we do so and consider some controls not applicable we always give a reasoning.

As a special case we treat aspects which would create infinite recursion, see example in chapter 3. For them, we do state internal applicability if we already stated external applicability for the same, respectively similar - due to the required adaptions for fitting a control for a management system to a CD system - requirement. However, we do not give any details on the implementability apart from assigning the aspect to the *Process*, because there is no point in doing so without repeating the same requirements of the external aspect reworded to the internal aspect. So every implementability cell in the section 6.1 which only states *Process* represents such a case.

The result of the assessment on applicability can be basically found in section 6.1.

6.1.2 Resulting SoA

The following SoA largely recites the ISO 27001[Isob], apart from the **Assignment**, **Implementability**, **Functional Requirement** and **Comment** columns. We consider this to be the clearest approach, even though it renders the table somewhat lengthy. Chapters 7 - Human Resource Security -, 11 - Physical and Environmental Security - and 15 - Supplier Relationships - are missing, since we consider them trivially not applicable (in accordance with a security expert, see Chapter 7).

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.5.1.1 - Policies for information security	A set of policies for information security shall be defined, approved by management, published and communicated to employees and relevant external parties	No, only definition of policy on organizational level			
		No, see external aspect			
A.5.1.2 - Review of the policies for information security	The policies for information security shall be reviewed at planned intervals or if significant changes occur to ensure their continuing suitability, adequacy and effectiveness	No, only review of policies (see A.5.1.1)			
		No, see external aspect			
A.6.1.1 - Information security roles and responsibilities	All information security responsibilities shall be defined and allocated	No, only definition of policies			However, responsibilities for the process, respectively process steps and for the assets like code, build-tooling, etc. has to be defined.
		No, see external aspect			
A.6.1.2 - Segregation of duties	Conflicting duties and areas of responsibility shall be segregated to reduce opportunities for unauthorized or unintentional modification or misuse of the organization's assets	No, nothing CD can provide			For roles and permissions: segregation of responsibilities
		No, see external aspect			
A.6.1.3 - Contact with authorities	Appropriate contacts with relevant authorities shall be maintained	No, nothing CD can support or provide			
		No, see external aspect			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.6.1.4 - Contact with special interest groups	Appropriate contacts with special interest groups or other specialist security forums and professional associations shall be maintained	No, same as A.6.1.3			
		No, see external aspect			
A.6.1.5 - Information security in project management	Information security shall be addressed in project management, regardless of the type of the project	Yes	Process: has to support mandatory building blocks in CD	System should restrict users in defining the CD process	Also requires the application of every security requirement which the CD system has to fulfil also for all built (internal) artefacts and systems
		Yes	Process		As a remark, information security shall also be addressed specifically in CD (before realization and individually for every organization) since it might affect the very same which includes security objectives and risk management (e.g. unauthorized access, usage of non-approved artefacts or non-availability)

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.6.2.1 - Mobile device policy	A policy and supporting security measures shall be adopted to manage the risks introduced by using mobile devices	No, has nothing to do with CD			
		No, see external aspect			
A.6.2.2 - Teleworking	A policy and supporting security measures shall be implemented to protect information accessed, processed or stored at teleworking sites	No, delivery system does not interact with teleworking sites			
		No, see external aspect			
A.8.1.1 - Inventory of assets	Assets associated with information and information processing facilities shall be identified and an inventory of these assets shall be drawn up and maintained	No, continuous delivery system does involve assets, however it is sufficient to assess them manually and on an organizational level			This should include software tools, build-nodes, the pipeline management system, information etc.
		No, see external aspect			
A.8.1.2 - Ownership of assets	Assets maintained in the inventory shall be owned	No, see 8.1.1			However, the definition of responsibility influences the roles and permissions assignment
		No, see 8.1.1			
A.8.1.3 - Acceptable use of assets	Rules for the acceptable use of information and of assets associated with information and information processing facilities shall be identified, documented and implemented	No, see 8.1.1			
		No, see 8.1.1			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.8.1.4 - Return of assets	All employees and external party users shall return all of the organizational assets in their possession upon termination of their employment, contract or agreement	No, no personnel responsibility of delivery system			
		No, see reasoning of external applicability			
A.8.2.1 - Classification of information	Information shall be classified in terms of legal requirements, value, criticality and sensitivity to unauthorised disclosure of modification	No, requirement does not require implementation of classification (see A.8.2.2)			As with A.8.1.1, this is more on an organizational level, but nonetheless should be done in general
		No, see external aspect			
A.8.2.2 - Labelling of information	An appropriate set of procedures for information labelling shall be developed and implemented in accordance with the information classification scheme adopted by the organization	No, no explicit labelling required, respectively useful. Should only be done on a much higher level than to classify every single produced artefact			The system could realize this by forcing the user to classify information / systems which is created during the process in the model. This has to follow the organizations classification scheme which could be checked
		No, see external aspect			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.8.2.3 - Handling of assets	Procedures for handling assets shall be developed and implemented in accordance with the information classification scheme adopted by the organization	No, see 8.1.1			
		No, see 8.1.1			
A.8.3.1 - Management of removable media	Procedures shall be implemented for the management of removable media in accordance with the classification scheme adopted by the organization	No, continuous delivery does not involve removable media			
		No, see external aspect			
A.8.3.2 - Disposal of media	Media shall be disposed of securely when no longer required, using formal procedures	No, delivery system involves no media			
		No, see external aspect			
A.8.3.3 - Physical media transfer	Media containing information shall be protected against unauthorized access, misuse or corruption during transportation	No, delivery system involves no media			
		No, see external aspect			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.9.1.1 - Access control policy	An access control policy shall be established, documented and reviewed based on business and information security requirements	Yes	System: Delivery systems need access control, which comprises authentication and the implementation of authorization. Additionally, records of significant events concerning use and management of user identities should be archived[Isoa]; Process: respecting authorization is responsibility of the process; Model: Privileged users have to assign permissions	All delivery systems should use version control system as only source of authentication; Delivery systems should archive records of significant events concerning the use and management of user identities; Permissions should be aligned across all delivery systems via a mapping. All delivery systems have to respect authorization based on the defined mapping and implement it as sole authorization measure	Is connected to A.8 in terms of roles vs access permissions on assets
		Yes	Process		NEEDINTINFO (access control policy)
A.9.1.2 - Access to networks and network services	Users shall only be provided with access to the network and network services that they have been specifically authorized to use	Yes	Process: same as A.9.1.1 only with special regard to network services (and with that to the process)	see A.9.1.1	
		Yes	Process		NEEDINTINFO (access control policy)

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.9.2.1 - User registration and de-registration	A formal user registration and de-registration process shall be implemented to enable assignment of access rights	Yes	System: delivery systems needs access control and thus registration and de-registration; Process: only allow unique user IDs, especially do not allow pseudo-accounts (shared accounts), respectively only allow them where they are necessary for business or operational reasons[Isoa]	Delegate registration / de-registration responsibility to company-wide system for authentication for (e.g. *Lightweight Directory Access Protocol (LDAP)*), thus version control system (which is used as source of authentication and authorization information, see A.9.1.1) has to support this system for authentication information; NEEDINTINF (mapping)	
		Yes	Process		
A.9.2.2 - User access provisioning	A formal user access provisioning process shall be implemented to assign or revoke access rights for all user types to all systems and services	Yes	Process: has to respect mapping which basically is the formal user access provisioning process	See A.9.1.1	
		Yes	Process		
A.9.2.3 - Management of privileged access rights	The allocation and use of privileged access rights shall be restricted and controlled	Yes	Process: see A.9.2.2; System: should control, respectively log privileged actions	System should log privileged actions; System should implement mapping (see A.9.1.1)	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.9.2.4 - Management of secret authentication information of users	The allocation of secret authentication information shall be controlled through a formal management process	Yes	Process: systems (test, development, etc.) are created during process thus process is responsible for respective secret authentication information allocation. This includes initial secure temporary secret authentication information which user's are forced to change on first use. Additionally, process should not continue prior of user acknowledging receipt of that authentication information; System: secret authentication information should be given to users in a secure manner	System has to implement process in this regard, namely creating temporary secret authentication information before creating any kind of system, forcing users to change it and acknowledge it before continuing the process. The temporary secret authentication information should be transferred in a secure manner to the respective user	
		Yes	Process		
A.9.2.5 - Review of user access rights	Asset owners shall review users' access rights at regular intervals	Yes	Process: responsible owners should be identified and documented; System: Inform respective owners regularly regarding all users' access rights. Log changes to privileged accounts	Build system should include owner information for created systems and has to provide functionality to regularly inform owners regarding user access rights. It has to log changes to privileged accounts	Use permission mapping for owner information
		Yes	Process		
A.9.2.6 - Removal or adjustment of access rights	The access rights of all employees and external party users to information and information processing facilities shall be removed upon termination of their employment, contract or agreement, or adjusted upon change	Yes	System: should disable user access in stated circumstances	Delegate this responsibility to the company-wide authentication system (e.g. LDAP), see A.9.2.1	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.9.3.1 - Use of secret authentication information	Users shall be required to follow the organization's practices in the use of secret authentication information	Yes	Process: is partly responsible for user's secret authentication information for created systems, thus process has to follow organization's practices. Furthermore, it has to force user to follow organization's practices on changing the password (see A.9.2.4); System: should keep secret authentication information confidential and ensure proper protection of passwords	System has to support process in creating respective authentication information and to restrict users in creating / using secret authentication information by themselves. System has to keep secret authentication information confidential and has to ensure proper protection of passwords	Realization could be done with the help of a companies password guideline; Also see A.9.2.4
		Yes	Process		
A.9.4.1 - Information access restriction	Access to information and application system functions shall be restricted in accordance with the access control policy	Yes	Process: has to respect mapping (see 9.1.1) which has to be fine granular enough to represent this requirement. This includes separating read, write, delete, and execute rights and specifically regarding access to build tools and created systems	System has to implement this mapping (see A.9.1.1); NEEDINTINFO (mapping, see A.9.1.1)	
		Yes	Process		check e.g. test systems for authentication
A.9.4.2 - Secure log-on procedures	Where required by the access control policy, access to systems and applications shall be controlled by a secure log-on procedure	Yes	System: log-on procedure should be secure. This includes all aspects of the ISO 27002[Isoa] listed as appendix in section 6.1.2	System has to implement a secure log-on procedure by including the stated requirements in section 6.1.2	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.9.4.3 - Password management system	Password management systems shall be interactive and shall ensure quality passwords	No, delivery systems perform no password management, only internal aspect			
		Yes	Process: has to check artefacts and created systems whether they perform password management and if so, check whether its interactive	System has to support process in checking the password management system	
A.9.4.4 - Use of privileged utility programs	The use of utility programs that might be capable of overriding system and application controls shall be restricted and tightly controlled	Yes	Process: has responsibility to restrict user in using utility programs, respectively only allowing users to perform a specified set of operations. This set might depend on authorization level of user; System: has to log usage of utility programs	System has to support process in restricting the set of operations. The system should be able to restrict the set of operations based on authorization levels of the user. The system has to log all usages of utility programs	
		Yes	Process		One could e.g. check for utility programs in created systems and their permissions
A.9.4.5 - Access control to program source code	Access to program source code shall be restricted	Yes	Process: all systems but the version control system should restrict access to source code (responsibility of mapping, see A.9.1.2). Version control system itself relies on manually set permissions	Delivery systems have to implement authorization based on mapping (see A.9.1.2)	
		Yes	Process		Only affects artefacts and systems whose functionality is version control

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.10.1.1 - Policy on the use of cryptographic controls	A policy on the use of cryptographic controls for protection of information shall be developed and implemented	No, only internal aspect			
		Yes	Process: force checks on that policy for every pipeline. Consider the artefacts and systems which are created	System has to support the ability to check artefacts and systems which are created for cryptographic controls (based on company policy) e.g. with compliance testing applications; NEEDINTINFO (company policy)	CD could help by checking whether only approved libraries and cryptographic algorithms are used or whether passwords and keys are stored in plaintext, etc.
A.10.1.2 - Key management	A policy on the use, protection and lifetime of cryptographic keys shall be developed and implemented through their whole lifecycle	No, see 10.1.1			
		Yes	Process: force checks on that policy for every pipeline. Consider the artefacts and systems which are created (see A.10.1.1)	System has to support the ability to check artefacts and systems which are created for this policy (see A.10.1.1); NEEDINTINFO (company policy)	

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.12.1.1 - Documented operating procedures	Operating procedures shall be documented and made available to all users who need them	No, only internal aspect, since this is not something the delivery system can provide			However, the duty of documentation still applies for all delivery systems; it is just not in the responsibility of the delivery system on a technical level
		Yes	Process: has to require operating procedures for systems which are created; Model: has to require URL where operating procedures can be found	System has to support process in requiring operating procedures. System has to support operating procedures URL in model.	
A.12.1.2 - Change management	Changes to the organization, business processes, information processing facilities and systems that affect information security shall be controlled	Yes	Process: has to include formal change procedure which among others comprises versioning, testing and informing on changes. Failing pipelines should have no effect. There should be no changes without version control; System: everything has to be versionable	System has to version source code, infrastructure as code, pipeline models, etc.. System has to implement two-man principle and integrate it in process which comprises support for multi-branch pipelines; System has to support process in informing and testing on changes; System has to support process regarding pipelines having no effect on failure; System has to force that there are no changes without version control. Systems should be as versionable as possible	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.12.1.3 - Capacity management	The use of resources shall be monitored, tuned and projections made of future capacity requirements to ensure the required system performance	Yes	System: has to monitor created systems. It also has to delete obsolete data	Pipeline management system has to monitor created systems. All delivery systems have to include self-cleaning (disk space) capabilities	
		Yes	Process		
A.12.1.4 - Separation of development, testing and operational environments	Development, testing, and operational environments shall be separated to reduce the risks of unauthorized access or changes to the operational environment	Yes	Process: release jobs should have audit logs. Process has to force the separation of development, testing and operational environments. Changes should be tested on testing or staging environments before being rolled out on operational environments. This has to follow rules of transfer from development to operational environments	System has to support process in providing audit logs for release jobs. System has to support process in enforcing separation	
		Yes	Process		
A.12.2.1 - Controls against malware	Detection, prevention and recovery controls to protect against malware shall be implemented, combined with appropriate user awareness	Yes	System: Apart from only allowing artefacts from internal repository manager, one may also allow whitelisted authorized web repositories;	System has to support restricting access to web repositories based on whitelists, respectively only allowing to use the internal repository manager	
		Yes	Process: has to include mandatory malware checks both for dependencies and for created systems. It has to force virus scanners on created systems	System has to provide malware scanners which process can use. System has to support process in forcing virus scanners on created systems	

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.12.3.1 - Information backup	Backup copies of information, software and system images shall be taken and tested regularly in accordance with an agreed backup policy	Yes	System: delivery system has to ease the creation of backups	System has to easy the creation of backups	Backups are not the responsibility of the delivery systems, however it has to easy its creation e.g. by pushing all deployed images to a central repository
		Yes	Process		
A.12.4.1 - Event logging	Event logs recording user activities, exceptions, faults and information security events shall be produced, kept and regularly reviewed	Yes	System: log respective aspects (see section 6.1.2)	System has to log respective aspects	
		Yes	Process		checking for logging dependencies or logging interfaces
A.12.4.2 - Protection of log information	Logging facilities and log information shall be protected against tampering and unauthorized access	Yes	Process: restricting access to log information has to be supported (and included in mapping, see A.9.1.1); System: has to prevent tampering of logs	All delivery systems have to protect log information against tampering. All delivery systems have to implement authorization based on permission mapping (as in A.9.1.2)	
		Yes	Process		
A.12.4.3 - Administrator and operator logs	System administrator and system operator activities shall be logged and the logs protected and regularly reviewed	Yes	System: log operator activities and protect them also	All delivery systems have to log operator activities and protect them also (see A.12.4.2)	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.12.4.4 - Clock synchronisation	The clocks of all relevant information processing systems within an organization or security domain shall be synchronised to a single reference time source	Yes	System: delivery systems should be synchronised to single reference time source	All delivery systems have to be synchronised to single reference time source	
		Yes	Process		
A.12.5.1 - Installation of software on operational systems	Procedures shall be implemented to control the installation of software on operational systems	Yes	System: everything including pipeline runs should be versioned (see A.12.1.2); Process: Artefacts which are released should be able to be linked to the pipeline run	Delivery systems have to version everything (see A.12.1.2) including pipeline runs. Pipeline runs have to be associated with the exact release name of an artefact	
		Yes	Process: has to restrict, respectively control installation of software on created operational systems	System has to support process in restricting this	We again need defined allocation of permissions
A.12.6.1 - Management of technical vulnerabilities	Information about technical vulnerabilities of information systems being used shall be obtained in a timely fashion, the organization's exposure to such vulnerabilities evaluated and appropriate measures taken to address the associated risk	No, only internal aspect			
		Yes	Process: include mandatory vulnerability analyses in the build of every software	System has to support respective scanners (like OWASP Dependency Check) and have them mandatorily integrated into pipelines	
A.12.6.2 - Restrictions on software installation	Rules governing the installation of software by users shall be established and implemented	No, only internal aspect			
		Yes	Process: has to restrict software which can be installed	System has to support process in restricting this	

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.12.7.1 - Information systems audit controls	Audit requirements and activities involving verification of operational systems shall be carefully planned and agreed to minimise disruptions to business processes	No, audit planning is not in the scope of CD			
		No, see external requirement			
A.13.1.1 - Network controls	Networks shall be managed and controlled to protect information in systems and applications	No, network is not responsibility of CD			
		No, see external requirement			
A.13.1.2 - Security of network services	Security mechanisms, service levels and management requirements of all network services shall be identified and included in network services agreements, whether these services are provided in-house or outsourced	No, see A.13.1.1			
		No, see external requirement			
A.13.1.3 - Segregation in networks	Groups of information services, users and information systems shall be segregated on networks	No, see A.13.1.1			However, CD should be separated network-wise. Also separate test- / demo-systems.
		No, see A.13.1.1			
A.13.2.1 - Information transfer policies and procedures	Formal transfer policies, procedures and controls shall be in place to protect the transfer of information through the use of all types of communication facilities	Yes	System, Process: have to support secure communication and follow those procedures	System has to allow for secure communication and has to support process	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.13.2.2 - Agreements on information transfer	Agreements shall address the secure transfer of business information between the organization and external parties	No, requirement only relevant for agreements			
		No, see external requirement			
A.13.2.3 - Electronic messaging	Information involved in electronic messaging shall be appropriately protected	Yes	System: whitelist allowed external communication. Protect allowed external communication	Delivery systems have to be able to function without external communication, respectively allow whitelisting of external communication which then should be appropriately protected	
		Yes	Process: ensure that created systems appropriately protect electronic messages; Model: user has to specify allowed external communication	System has to enforce user specified whitelist with a firewall based on *Internet Protocol (IP)* addresses or with a proxy based on domain names, respectively *Uniform Resource Locators (URLs)*. System should only allow *Transport Layer Security (TLS)* connections.	
A.13.2.4 - Confidentiality or non-disclosure agreements	Requirements for confidentiality or non-disclosure agreements reflecting the organization's needs for the protection of information shall be identified, regularly reviewed and documented	No, requirement only relevant for those agreements			
		No, see external requirement			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.14.1.1 - Information security requirements analysis and specification	The information security related requirements shall be included in the requirements for new information systems or enhancements to existing information systems	No, only internal aspect			
		Yes	Process: ensures that all information security related requirements are always included / checked for all created systems	System has to support process in checking this, respectively implement checking functionality	Should be done once for CD
A.14.1.2 - Securing application services on public networks	Information involved in application services passing over public networks shall be protected from fraudulent activity, contract dispute and unauthorized disclosure and modification	No, same as A.14.1.1			
		Yes	Process: check for protection of respective information	Support process in checking protection of information involved in application services passing over public networks	
A.14.1.3 - Protection application services transactions	Information involved in application service transactions shall be protected to prevent incomplete transmission, mis-routing, unauthorized message alteration, unauthorized disclosure, unauthorized message duplication or replay	No, only internal aspect			
		Yes	Process: check for protection of information involved in application service transactions	System has to support process in checking protection of information involved in application service transactions	

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.14.2.1 - Secure development policy	Rules for the development of software and systems shall be established and applied to developments within the organization	Yes	Process: all systems should communicate with version control system in a secure manner*ISO 27002*[Isoa]; System: *ISO 27002*states security in the version control system, especially considering secure repositories	System has to support process in communicating with version control system in a secure manner as well as ensuring that the version control system is secure including its repositories	
		Yes	Process: has to enforce those rules on created systems and artefacts	Support process in enforcing rules regarding development of software and systems	
A.14.2.2 - System change control procedures	Changes to systems within the development lifecycle shall be controlled by the use of formal change control procedures	Yes	Process: has to integrate review steps. Process should check whether every commit is associated with a case / ticket	System has to support process in forcing review steps. System has to support process in checking whether every commit is associated with a case / ticket	
		No, only external aspect			
A.14.2.3 - Technical review of applications after operating platform changes	When operating platforms are changed, business critical applications shall be reviewed and tested to ensure there is no adverse impact on organizational operations or security	Yes	Process: on operating platform changes, pipelines should be triggered and all tests should be run again	System has to support triggering of pipelines in general and especially should support triggers based on operating platform changes	
		Yes	Process		
A.14.2.4 - Restrictions on changes to software packages	Modifications to software packages shall be discouraged, limited to necessary changes and all changes shall be strictly controlled	Yes	System: should monitor modifications to 3rd party software	Delivery systems have to monitor the use of modified 3rd party software	
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.14.2.5 - Secure system engineering principles	Principles for engineering secure systems shall be established, documented, maintained and applied to any information system implementation efforts	No, only internal aspect			
		Yes	Process: has to integrate checks in order to ensure that created systems comply with those principles	System has to support process in checking this, respectively implement checking functionality	
A.14.2.6 - Secure development environment	Organizations shall establish and appropriately protect secure development environments for system development and integration efforts that cover the entire system development lifecycle	No, development environment is not responsibility of delivery system			However, has to be also considered for CD itself; however, this is nothing the delivery system has to provide
		No, see external reasoning including comment			
A.14.2.7 - Outsourced development	The organization shall supervise and monitor the activity of outsourced system development	Yes	Process: same measures which are applied for internal development should be applied to outsourced development. If this is not possible, the use of external artefacts should be closely monitored	System has to support monitoring of external artefacts.	
		Yes	Process		
A.14.2.8 - System security testing	Testing of security functionality shall be carried out during development	No, only internal aspect			
		Yes	Process: has to include tests, respectively verify the existence of tests on security functionality	System has to support process in checking for security functionality tests	

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.14.2.9 - System acceptance testing	Acceptance testing programs and related criteria shall be established for new information systems, upgrades and new versions	No, only internal aspect			
		Yes	Process: has to enforce acceptance testing programs and related criteria	System has to support process in enforcing acceptance testing programs and related criteria	also for outsourced development (see A.14.2.7)
A.14.3.1 - Protection of test data	Test data shall be selected carefully, protected and controlled	No, delivery system has no influence on test data selection			
		No, see external aspect			Since every created system restricts allowed outgoing connections, user has to explicitly specify test data if it is gathered from an external resource
A.16.1.1 - Responsibilities and procedures	Management responsibilities and procedures shall be established to ensure a quick, effective and orderly response to information security incidents	No, delivery system is not responsible for management procedures			It is part of them and has to support them though
		No, see external aspect			
A.16.1.2 - Reporting information security events	Information security events shall be reported through appropriate management channels as quickly as possible	Yes	Process: report respective events	System has to support process in reporting respective events (e.g. style checks, cryptography, use of libraries) to management	Events to be considered comprise unauthorized access, discovery of vulnerabilities, plain text passwords
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.16.1.3 - Reporting information security weaknesses	Employees and contractors using the organization's information systems and services shall be required to note and report any observed or suspected information security weaknesses in systems or services	No, delivery system has no personnel responsibility			
		No, see external requirement			
A.16.1.4 - Assessment of and decision on information security events	Information security events shall be assessed and it shall be decided if they are to be classified as information security incidents	Yes	Process: has to assess security events based on defined metrics in order to classify them	System has to support assessment of security events	
		Yes	Process		
A.16.1.5 - Response to information security incidents	Information security incidents shall be responded to in accordance with the documented procedures	No, impossible for delivery system to detect incident			Procedure have to be documented though
		No, see external aspect			
A.16.1.6 - Learning from information security incidents	Knowledge gained from analysing and resolving information security incidents shall be used to reduce the likelihood or impact of future incidents	No, see A.16.1.5			
		No, see external requirement			

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.16.1.7 - Collection of evidence	The organization shall define and apply procedures for the identification, collection, acquisition and preservation of information, which can serve as evidence	Yes	Process: has to log everything, especially everything related to security incidents	System has to support this comprehensive logging	
		Yes	Process		
A.17.1.1 - Planning information security continuity	The organization shall determine its requirements for information security and the continuity of information security management in adverse situations, e.g. during a crisis or disaster	No, only policy			
		No, see external aspect			
A.17.1.2 - Implementing information security continuity	The organization shall establish, document, implement and maintain processes, procedures and controls to ensure the required level of continuity for information security during an adverse situation	No, delivery system is not specifically responsible in adverse situations			
		No, see external aspect			
A.17.1.3 - Verify, review and evaluate information security continuity	The organization shall verify the established and implemented information security continuity controls at regular intervals in order to ensure that they are valid and effective during adverse situations	No, delivery system has no specific responsibility regarding adverse situations			
		No, see external aspect			
A.17.2.1 - Availability of information processing facilities	Information processing facilities shall be implemented with redundancy sufficient to meet availability requirements	Yes	System: delivery systems shall be redundant	Systems have to be able to be run redundant	However, availability requirements have to be checked before
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.18.1.1 - Identification of applicable legislation and contractual requirements	All relevant legislative statutory, regulatory, contractual requirements and the organization's approach to meet these requirements shall be explicitly identified, documented and kept up to date for each information system and the organization	No, not the responsibility of the delivery system			
		No, see external aspect			
A.18.1.2 - Intellectual property rights	Appropriate procedures shall be implemented to ensure compliance with legislative, regulatory and contractual requirements related to intellectual property rights and use of proprietary software products	No, only internal aspect			
		Yes	Process: has to integrate respective scanners to ensure compliance with legislative, regulatory and contractual requirements	System has to support process in ensuring compliance with legislative, regulatory and contractual requirements	
A.18.1.3 - Protection of records	Records shall be protected from loss, destruction, falsification, unauthorized access and unauthorized release, in accordance with legislatory, regulatory, contractual and business requirements	Yes	System: has to deal with preservation of records	System has to protect records from loss, destruction, falsification, unauthorized access and unauthorized release	
		Yes	Process		
A.18.1.4 - Privacy and protection of personally identifiable information	Privacy and protection of personally identifiable information shall be ensured as required in relevant legislation and regulation where applicable	Yes	System, Process: every action in CD is personally identifiable information and have to be protected	System has to protect and support process in protecting logs of actions of delivery systems	Additionally, after a certain period of time, this information has to be anonymized
		Yes	Process		

ID	Control	**Applicability**	**Implementability**	**Functional Requirement**	**Comment**
A.18.1.5 - Regulation of cryptographic controls	Cryptographic controls shall be used in compliance with all relevant agreements, legislation and regulations	No, delivery system has no responsibility over cryptographic controls; it can only implement them			
		No, see external aspect			
A.18.2.1 - Independent review of information security	The organization's approach to managing information security and its implementation (i.e. control objectives, controls, policies, processes and procedures for information security) shall be reviewed independently at planned intervals or when significant changes occur	No, requires human interaction			
		No, see external aspect			
A.18.2.2 - Compliance with security policies and standards	Managers shall regularly review the compliance of information processing and procedures within their area of responsibility with the appropriate security policies, standards and any other security requirements	No, delivery system has no personnel responsibility			
		No, see external aspect			
A.18.2.3 - Technical compliance review	Information systems shall be regularly reviewed for compliance with the organization's information security policies and standards	No, this is more on an organizational level			
		No, see external aspect			

Appendix for A.9.4.2

- not display system or application identifiers until the log-on process has been successfully completed
- display a general notice warning that the computer should only be accessed by authorized users
- not provide help messages during the log-on procedure that would aid an unauthorized user
- validate the log-on information only on completion of all input data. If an error condition arises, the system should not indicate which part of the data is correct or incorrect
- protect against brute force log-on attempts
- log unsuccessful and successful attempts
- raise a security event if a potential attempted or successful breach of log-on controls is detected
- display the following information on completion of a successful log-on:
- not display a password being entered
- not transmit passwords in clear text over a network
- terminate inactive sessions after a defined period of inactivity, especially in high risk locations such as public or external areas outside the organization's security management or on mobile devices
- restrict connection times to provide additional security for high-risk applications and reduce the window of opportunity for unauthorized access. Distinguish between interactive and automatic sessions

Appendix for A.12.4.1

Event logs should include, when relevant:

- user IDs
- system activities
- dates, times and details of key events, e.g. log-on and log-off
- device identify or location if possible and system identifier
- records of successful and rejected system access attempts
- records of successful and rejected data and other resource access attempts
- changes to system configuration
- use of privileges
- use of system utilities and applications

- files accessed and the kind of access
- network addresses and protocols
- alarms raised by the access control system
- activation and de-activation of protection systems, such as anti-virus systems and detection systems
- records of transactions executed by users in applications

6.1.3 Assignment Statistics

Table 6.2 shows basically two important information. The first one is the number of applicable controls in contrast to the number of total controls per chapter. The second one is the number of assignments to model, process or system. We mainly consider the external aspects and put the numbers regarding the internal aspects in brackets. With *externally* and *internally* we refer to the requirements the delivery systems have to fulfil and the requirements the artefacts and systems have to fulfil. There might be multiple assignments per control.

Section	Number of Controls	Applicable	Model	Process	System
5	2	0 (0)	0 (0)	0 (0)	0 (0)
6	7	1 (1)	0 (0)	1 (1)	0 (0)
7	6	0 (0)	0 (0)	0 (0)	0 (0)
8	10	0 (0)	0 (0)	0 (0)	0 (0)
9	14	13 (14)	1 (0)	11 (14)	8 (0)
10	2	0 (2)	0 (0)	0 (2)	0 (0)
11	15	0 (0)	0 (0)	0 (0)	0 (0)
12	14	10 (13)	0 (1)	4 (13)	9 (0)
13	7	2 (2)	0 (1)	1 (2)	2 (0)
14	13	5 (10)	0 (0)	4 (10)	2 (0)
15	5	0 (0)	0 (0)	0 (0)	0 (0)
16	7	3 (3)	0 (0)	3 (3)	0 (0)
17	4	1 (1)	0 (0)	0 (1)	1 (0)
18	8	2 (3)	0 (0)	1 (3)	2 (0)
All / Total	114	37 (49)	1 (2)	25 (49)	24 (0)

Table 6.2: Number of applicable controls of ISO 27001 (27002) with assignment to model, system or process

The first thing we can observe is that there are only three *model* related requirements in total. Regarding the relation between *process*-related and

system-related requirements, we see that there are 25 *external process*-related requirements versus 24 *external system*-related requirements which means they are quite balanced. For the *internal* view, we have no *system*-related requirements, but 49 *process*-related requirements.

Other than that, of the 114 controls of the ISO, 37 are applicable *externally* and 49 *internally*. Thus, all *internally* applicable aspects are *process*-related, where two of them also require user-input in form of them also being *model*-related. In contrast to that, the 37 *externally* applicable aspects have more overlapping assignments, since there are 49 assignments to *process* or *system*.

We mentioned that one can see that there is no direct responsibility of the *system* for any of the requirements imposed for artefacts and systems. However, as seen in section 6.4, internal requirements may have dependencies to requirements which are the responsibility of the system.

6.2 Resulting Major Categories

Based on the results and concepts of section 6.1, we identified the following major categories (subsections) regarding the ISO 27001 which a delivery system can or has to implement, respectively realize in order to fulfil the ISO 27001. For each category, we list the major concepts (functional requirements) which have to be realized in that category. Those major concepts are based on the detailed aspects of the SoA (see section 6.1.). Notably, we summarize aspects and also omit some concepts which we identified as trivial functional requirements in order to gain an overview of concepts in every category. The result is basically a summary of the concepts in section 6.1 assigned to the categories we identified. To have better traceability and since the following requirements are the basis of our models, we annotated all those requirements with a letter whose related control has multiple different requirements.

The following catalogue raises no claim to completeness, every single requirement can be found in the SoA (see section 6.1). Furthermore, the category names resemble or are identical to sections of the ISO 27001, however, they do not necessary deal with the aspects which are listed under the respective sections in the ISO 27001.

Based on these major categories, we can - for the realization concepts in these categories - apply them to a varying extend which is shown the realization roadmap in section 6.3.

6.2.1 1. Access Management

- A.9.1.1(a) - use of version control system as source of authentication (credentials) for rest of delivery system

- A.9.1.1(b), A.9.1.2, A.9.2.2, A.9.2.3(a), A.9.4.1, A.9.4.5, 12.4.2(a) - user authentication information mapping between version control system and rest of delivery system
- A.9.2.1 - version control system has to support authentication via company-wide system for authentication (e.g. LDAP)
- A.9.2.4 - temporary secret authentication information for created systems including forcing the user to change and acknowledge it
- A.9.2.5(a) - owner information for created systems and regularly inform them about user access rights
- A.9.3.1(a) - creation of authentication information according to company policies and forcing users changing it to follow company policies
- A.9.3.1(b) - ensure proper protection of passwords and keep authentication information confidential
- A.9.4.2 - functional requirements regarding the detailed implementation of the access management (login) system
- A.9.4.4(a) - System has to support restricting modifications of the process based on authorization levels
- A.12.2.1(a) - software only through internal artefact storage, respectively whitelisted web repositories
- A.13.2.3(a) - whitelist and protect allowed external communication

6.2.2 2. Logging and Monitoring

- A.9.1.1(c) - log significant events concerning the use and management of user identities
- A.9.2.3(b) - log privileged actions
- A.9.2.5(b) - log changes to privileged accounts
- A.9.4.4(b) - log all usages of utility programs
- A.12.1.3(a) - monitoring of created systems
- A.12.1.4(a) - release jobs should have audit log
- A.12.4.1 - log all aspects stated in A.12.4.1
- A.12.4.2(b), A.12.4.3(a) - Protect log against alterations, deletion, etc.; also consider possible manipulation by privileged users
- A.12.4.3(b) - log operator activities
- A.14.2.4 - monitoring of the use of modified 3rd party software
- A.14.2.7 - monitoring of external artefacts
- A.16.1.2 - report information security events, e.g. access violations, etc.
- A.16.1.7 - log everything
- A.18.1.3 - protection of records (loss, destruction, etc.)
- A.18.1.4 - protection of personally identifiable information in activity logs

6.2.3 3. Change Management

- A.12.1.2(a) - deal with failing pipelines in a respective manner (no deployment, etc.)
- A.12.1.2(b) - multiple branches support for e.g. separate release jobs including forcing of two-man principle
- A.12.1.2(c), A.12.5.1(a) - no changes without version control: version everything including pipeline runs, models, system configuration
- A.12.1.2(d), A.14.2.3 - support triggering of pipelines (testing and informing) on changes to pipelines, code, operating platform changes
- A.12.1.2(e) - Systems shall be as versionable as possible
- A.12.1.4(b) - force separation of development, testing and operational environments
- A.12.1.4(c) - rules for transfer of software from development to operational status
- A.12.1.4(d) - changes to operational systems should be tested first in a testing or staging environment
- A.12.5.1(b) - release pipeline runs have to be associated with the exact release name of an artefact
- A.14.2.2(a) - integrate review steps
- A.14.2.2(b) - check whether every commit is linked to case

6.2.4 4. Artefacts and Systems (ONLY internal aspects)

- A.10.1.1, A.10.1.2 - checks regarding cryptographic policies including key management, etc.
- A.12.1.1 - require operating procedures (by specifying URL)
- A.12.2.1(b) - integrate malware checks for dependencies and created systems
- A.12.2.1(c) - force virus scanners on created systems
- A.12.5.1(c) - control installation of software on operational systems
- A.12.6.1 - mandatory vulnerability analyses for every software build
- A.13.2.3(b) - ensure that created systems appropriately protect electronic messages
- A.14.1.1 - ensure that all information security related requirements are always checked for all create systems
- A.14.1.2 - check for protection of information involved in application services passing over public networks
- A.14.1.3 - check for protection of information involved in application service transactions
- A.14.2.1(a) - checks for secure coding guideline compliance
- A.14.2.5 - obey principles of secure system engineering
- A.14.2.8 - tests for security functionality
- A.14.2.9 - enforce acceptance testing

- A.18.1.2 - ensure compliance with legislative, regulatory and contractual requirements

6.2.5 5. Other Policies/Requirements

- A.6.1.5 - mandatory building blocks in CD process, as well as restrictions for users
- A.12.1.3(b) - deletion of obsolete data, e.g. old snapshots, build data, etc.
- A.12.3.1 - ease creation of backups
- A.12.4.4 - clock synchronisation
- A.13.2.1 - support, respectively allow secure communication
- A.14.2.1(b) - security in the version control including secure communication with version control system and secure repositories
- A.16.1.4 - assessment of security events
- A.17.2.1 - delivery system should be redundant

6.3 Realization Roadmap

In section 6.2, we created 5 categories with all requirements. Based on those requirements and categories, we create the following realization roadmap.

First, we introduce three levels. The levels are ordered according to the effort the aspects contained in them require to be implemented. They range from *minimal effort* over *medium effort* to *high effort*.

Each of the aspects in section 6.2 can be assigned to one or (by splitting them) more levels and we further abstract them to fit into the model. For reasons of clarity and visibility, we split the model by columns hence having five tables (see Table 6.3, Table 6.4, Table 6.5, Table 6.6, Table 6.7).

The category *3. Change Management* is quite closely related to CD and thus is somewhat aligned to it, meaning that the level of automation required increases per required level of effort.

Levels / Areas	1. Access Management
High effort	System Access: • A.9.2.4, 9.3.1(a) • A.9.2.5(a) • A.9.3.1(b) • A.9.4.2 • A.9.4.4(a)
Medium effort	Authorization: • A.9.1.1(b), A.9.1.2, A.9.2.2, A.9.2.3(a), A.9.4.1, A.9.4.5, 12.4.2(a) • A.12.2.1(a) • A.13.2.3(a)
Minimal effort	Authentication: • A.9.1.1(a) • A.9.2.1

Table 6.3: Realization roadmap *Access Management*

Levels / Areas	2. Logging and Monitoring
High effort	Protection: • A.12.4.2(b), A.12.4.3(a), A.18.1.3 • A.18.1.4
Medium effort	Reporting: • A.16.1.2 • A.12.1.4(a)
Minimal effort	Information: • A.9.1.1(c), A.9.2.3(b), A.9.2.5(b), A.9.4.4(b), A.12.4.1, A.12.4.3(b), A.16.1.7 • A.12.1.3(a) • A.14.2.4 • A.14.2.7

Table 6.4: Realization roadmap *Logging and Monitoring*

Levels / Areas	3. Change Management
High effort	System-reliant change management: • A.12.1.2(d), A.14.2.3 • A.12.1.2(e)
Medium effort	Process-reliant change management: • A.12.1.2(a) • A.12.1.2(b) • A.12.5.1(b) • A.14.2.2(a) • A.14.2.2(b) • A.12.1.4(b) • A.12.1.4(c) • A.12.1.4(d)
Minimal effort	Minimal change management: • A.12.1.2(c), A.12.5.1(a)

Table 6.5: Realization Roadmap *Change Management*

Levels / Areas	4. Artefacts and Systems
High effort	Harder automatable process steps: • A.12.5.1(c) • A.14.2.8 • A.14.2.9 • A.18.1.2
Medium effort	Easily automatable process steps: • A.12.1.1 • A.12.2.1(c)
Minimal effort	Scanners: • A.10.1.1, A.10.1.2 • A.12.2.1(b) • A.12.6.1 • A.13.2.3(b) • A.14.1.1 • A.14.1.2 • A.14.1.3 • A.14.2.1(a) • A.14.2.5

Table 6.6: Realization Roadmap *Artefacts and Systems*

Levels / Areas	5. Other Policies
High effort	• A.6.1.5 • A.13.2.1 • A.16.1.4
Medium effort	• A.12.1.3(b) • A.14.2.1(b) • A.17.2.1
Minimal effort	• A.12.3.1 • A.12.4.4

Table 6.7: Realization Roadmap *Other Policies*

6.4 Dependency Model

In order to be able to prioritize single controls and to get a roadmap not only based on estimated effort (see section 6.3), we created a dependency model shown in Figure 6.1.

It consists out of all requirements which can be found in section 6.2. Every requirement which depends on another requirement has an outgoing arrow to the respective requirement. Thereby, a requirement A depends on a requirement B if requirement B has to be realized before requirement A can be realized. Additionally, the requirements are coloured according to their assignment in the SoA. If a requirement has multiple assignments it is coloured with both colours (see legend). Moreover, there are two different kind of arrow lines. Dashed arrow lines and solid arrow lines. The solid arrow lines describe a dependency relation where all requirements the arrow points to have to be fulfilled before the other one can be realized. In contrast, dashed lines mean that only one of the requirements the dashed arrow points to has to be realized.

In order to group the requirements and to keep the model clear, we adopted the categories identified in section 6.2. Additionally, this grouping clearly shows all *internal* aspects as they can all be found in *4. Artefacts and Systems*. We shortly discuss this assessment in the context of the resulting maturity model (see section 6.5) in section 7.3.

All in all, we can see that there are only 5 inter-category dependencies from which four of them depend on requirement A.6.1.5. Other than that, category one and two have the most intra-category dependencies.

6.5 Maturity Model

In accordance with our goal to provide means for a company to strategically implement the ISO 27001, we created a maturity model on the basis of our dependency model (see section 6.4). This means that every maturity level above the first one requires some control(s) in the lower maturity level to be fulfilled which happens to be one characteristic of a maturity model. We basically extracted the different hierarchies of the dependency model into this maturity model.

The result is shown in Figure 6.2. It consists out of three maturity levels. The lowest level is the *System-oriented* level meaning all aspects require some actions in the system. The next level is the *System-dependent* level where the majority of requirements are *system* related requirements and require adaptions to the system. Finally, the highest level is the *Process-oriented* level where most aspects require adaptions to the process.

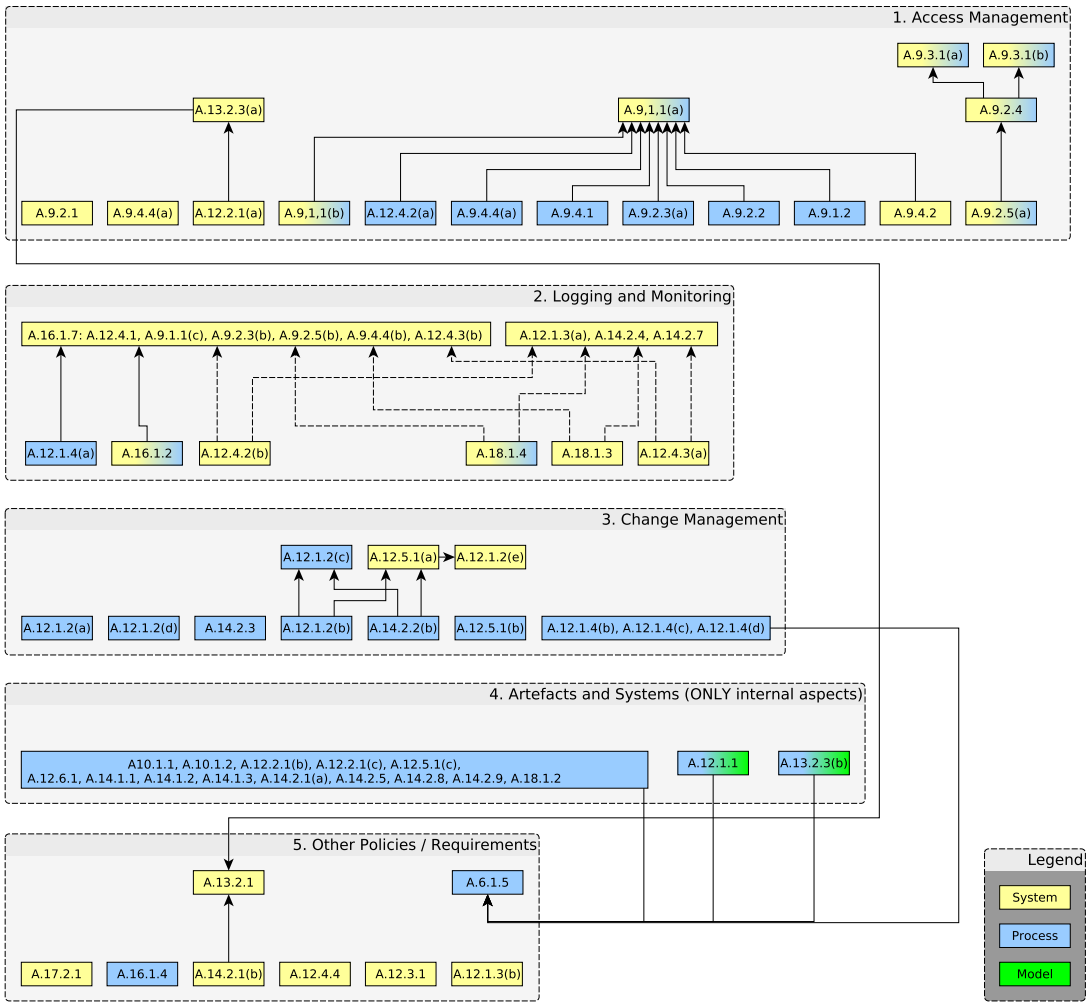

Figure 6.1: Dependencies between requirements of ISO 27001

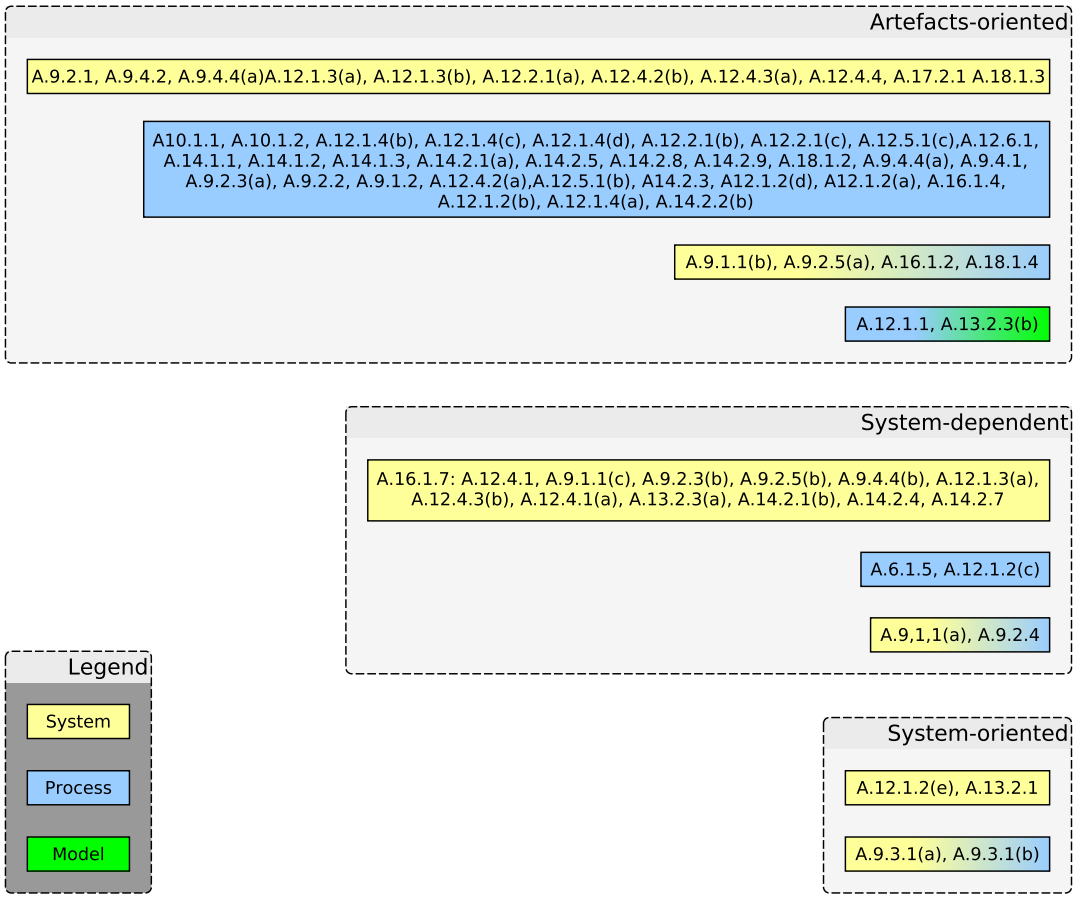

Figure 6.2: Maturity model as result of dependency model

A discussion of our maturity model can be found in chapter 7.

6.6 CD maturity meets CD ISO Maturity Model

In this section, we deal with the mapping of our maturity model from section 6.5 to the CD maturity model shown in Figure 2.1, respectively section 2.2.1. For this mapping, we take every single requirement listed in section 6.2 and assign it to a cell of the maturity model shown in Figure 2.1. Therefore, if multiple assignments are possible, we assign the requirement to the cell requiring the highest maturity. Assignment here is defined as either a requirement can be found similarly in Figure 2.1 or a requirement depends on an aspect in Figure 2.1.

In the following we shortly discuss some difficult to assign requirements and give our reasoning which might also serve as an example, respectively guidance for the other aspects we assigned.

- A.9.1.1(a): Authentication itself is not really dealt with in the CD maturity model. However, we consider all authentication and authorization requirements to be mappable to a *Standard process for all environments*. So this also includes requirements A.9.1.1(b), A.9.1.2, A.9.2.2, etc.
- A.9.2.4: Also requires a standard process for all environments, such that all systems are created with temporary secret authentication information
- A.9.2.5(a): We map component ownership to ownership of systems and artefacts
- A.12.1.3(a): We map this also to *Standard process for all environments* since no other aspect is quite fitting and we need a standard process for all environments which includes monitoring of created environments
- A.14.2.4, A.14.2.7: In contrast to A.12.1.3(a), those two requirements are elements of the process and thus mapped to *Measure the process*. A.12.1.3 is more something the process has to enforce
- A.18.1.3, A.18.1.4: Both aspects were are mapped to *Report History Available* as closest match. In section 6.7 we only consider protection of personally identifiable information in historic reports
- A.12.1.2(c), A.12.5.1(a): Infrastructure as code required for versioning infrastructure
- A.18.1.2: Compliance testing is not listed in the maturity model. Eventually, we mapped it to the *Advanced* level, because this level contains all fully automated testing aspects

The resulting assignment can be seen in Figure 6.3 and can be compared to Figure 2.1 (side-by-side). The order of the entries is one after another according to the order of the requirements in section 6.2.

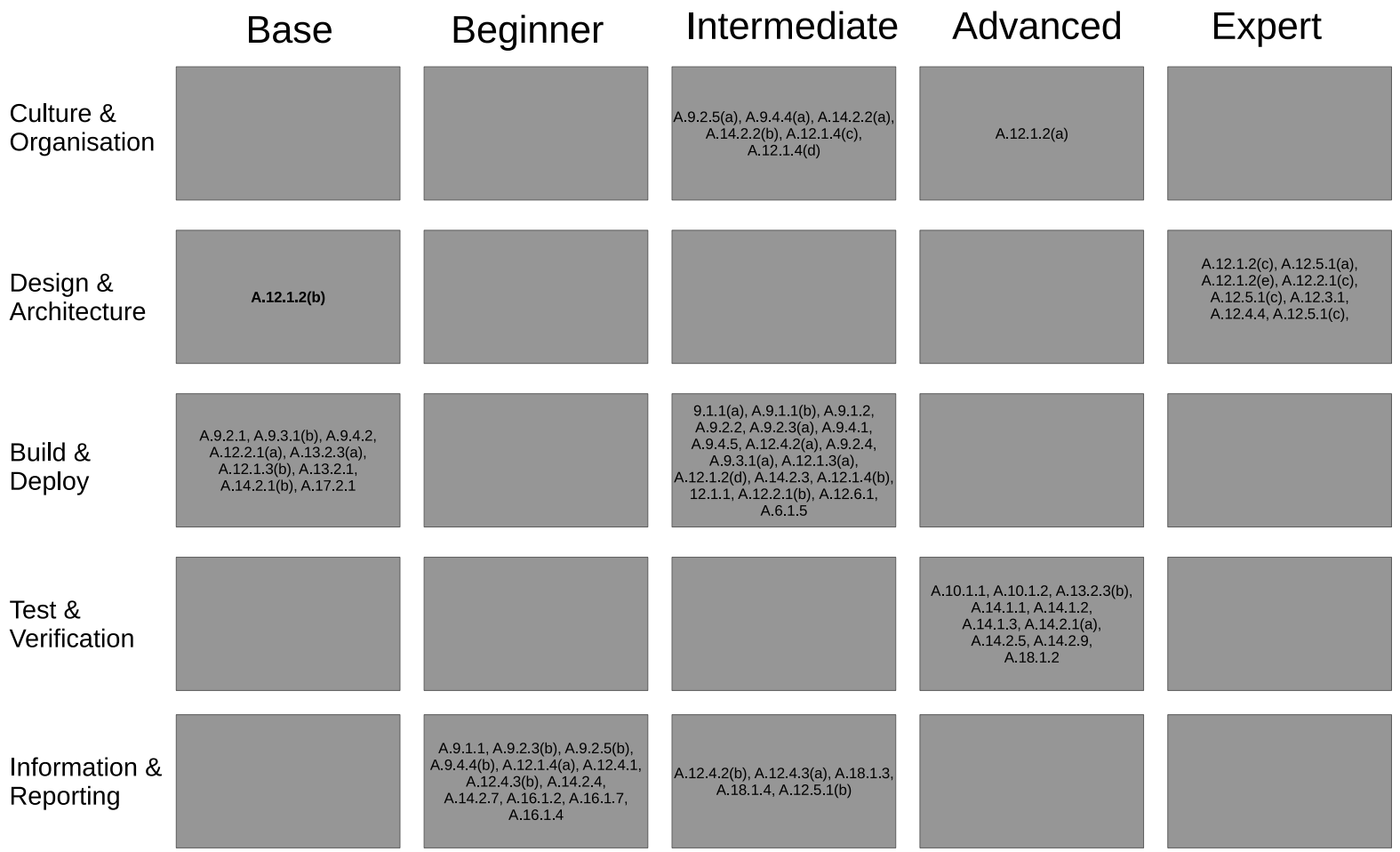

Figure 6.3: Mapping of CD maturity model with our security maturity model

After all, difficult to assign are those aspects which can be realized to a varying degree. We had to e.g. weigh up whether IaC was more relevant to assign the requirement or some other aspect as many requirements can only be realized fully automated if we use IaC. In general, most of our extracted requirement could be assigned to different maturity levels. An example for this is *A.16.1.2 - report information security events, e.g. access violations, etc.*. It could be realized with *manual reporting* in which case it would belong to the *Information & Reporting Base* level, but also with *Scheduled Quality Reports* in which case it would belong to the *Beginner* level of the same category. In the end, this is one of the reasons why we chose to always assign the requirement to the cell holding the highest maturity level if multiple assignments are possible.

Interesting to see is that the model has some *hotspots* where many requirements are mapped, respectively assigned to. Those comprise *versioned code base*, *dedicated build server*, *measure the process*, *common process for all changes*, *standard process for all environments*, *report history is available*, *full automatic tests*, *infrastructure as code*. Strikingly those fit exactly to our categories (see section 6.2) *2. Logging and Monitoring*, *3. Change Management* and *4. Artefacts and Systems* (regarding tests).

Furthermore, we see that on average an *Intermediate* level of maturity is required for security, whereby eight aspects require, respectively are mapped to IaC which is assigned to *Expert* level maturity by Rehn et al.[BRP13].

Finally, a special case is A.12.1.2(b), since it basically stands against CD. The CD maturity model attests a higher maturity for fewer branches whereas A.12.1.2(b) requires multi-branch support. Thus, we assigned it to the *Base* level.

6.7 Realization Suggestions

After having done the assignment and categorization in section 6.2, for each category, we determine concrete realizations based on the functional requirements. At that, we aim for maximising the degree of automation. However, here *realization* has to be understood as a collection of realization concepts, respectively suggestions for the concrete environment (see section 6.7.1) which then has to be applied step-by-step to a production environment (future work, respectively not scope of this thesis). It is only intended to give guidance on how a possible realization might look like and to get an idea whether the requirements are realizable at all. Eventually, this Chapter is also shortly considered in the discussion.

6.7.1 Environment

In order to be able to concretise all aspects, we have to define our environment. For simplicity we always assume the latest versions (as of September 2019).

- Gitlab - only for version control, not for CD features / pipeline management
- Nexus
- Jenkins
- Maven
- Java

Limiting the environment to the above *technologies* means that there is a high possibility that some of the proposed realization ideas might not work for other technologies or that there do not even exist any realization. However, this is not the scope of this chapter. We only want to give realization ideas to get a better understanding of the SoA and to get an idea to which degree the ISO 27001 is realizable. The links given were accessed on September 2019 where most of them originate either from [Son], [Jen], [Git] or [OWA].

6.7.2 1. Access Management

A.9.1.1(a)

Use Gitlab OAuth Plugin for Jenkins[1] and use similar plugin[2] for Nexus.

Both plugins allow to use Gitlab as authentication provider.

A.9.1.1(b), A.9.1.2, A.9.2.2, A.9.2.3(a), A.9.4.1, A.9.4.5, 12.4.2(a)

For Jenkins, use the "Use Gitlab repository permissions" feature to map user permissions from Gitlab to Jenkins.

Assuming a perfectly realised continuous delivery process, only Jenkins should have deployments rights to the Nexus and every authenticated user shall have read access.

A.9.2.1

Gitlab supports authentication via LDAP[3]

[1] https://wiki.jenkins.io/display/JENKINS/Gitlab+OAuth+Plugin
[2] https://github.com/auchanretailfrance/nexus3-gitlabauth-plugin
[3] https://docs.gitlab.com/ee/administration/auth/ldap.html

A.9.2.4

Enforcing the change of user credentials on next login can be done with *Operating System (OS)* means. For e.g. Windows via *wmic*, and *wmic* and for e.g. Linux with *chage*.

A.9.2.5(a)

Use vSphere plugin (see A.12.1.3(a)); Set owner information in notes field (of vSphere) via the *Reconfigure VM* step. Owner information therefore is needed in pipeline, otherwise pipeline should fail. Can be done in pipeline *Domain Specific Language (DSL)*. Regarding information of owners create extra jobs.

A.9.3.1(a)

Implement creation of password in pipeline DSL according to company policies. Create *Virtual Machine (VM)* templates (see A.9.2.5(a), A.12.1.3(a)) which force the company's password policy as system password policy.

A.9.3.1(b)

All systems encrypt authentication information.

A.9.4.2

Since we intend to use Gitlab as authentication provider for Jenkins, we also get there the Gitlab login screen which out-of-the-box does support all but two of the aspects. The first not requirement not supported out-of-the-box is the warning message that the computer should only be accessed by authorized users. This warning can be added via a branded login page[4]. Unfortunately, there is no easy way to enforce the restriction of the connection times.

Regarding Nexus, if anonymous access is disabled it supports, respectively not supports the same aspects as Gitlab. Restricting the connection times is not possible either, but in contrast to Gitlab, creating a warning notice does not seem to be possible.

A.9.4.4(a)

Gitlab does not offer permissions granular enough to provide such restrictions. Either a user has the access right to modify the process or he does not.

[4] `https://docs.gitlab.com/ee/customization/branded_login_page.html`

A.12.2.1(a)

This only affects Jenkins, which is considered in A.13.2.3(a).

A.13.2.3(a)

The easiest way to prevent all unwanted external communication is to use an application firewall on the system (e.g. modify the hosts file in order to blacklist all Top-Level-Domains and whitelist the allowed domains; even though the requirement requires whitelisting, the technical realization first requires blacklisting). For Jenkins, only access to Gitlab and Nexus should be granted. Depending on the scanners needed, we might also have to whitelist connections to online databases (such as CVE). Gitlab should not be allowed any external communication. However, Nexus requires communication to the whitelisted repositories.

6.7.3 2. Logging and Monitoring

A.9.1.1(c)

Use logging facilities provided by all three applications (Jenkins[5], Nexus[6], Gitlab[7]).

A.9.2.3(b)

See A.9.1.1(c).

A.9.2.5(b)

See A.9.1.1(c).

A.9.4.4(b)

See A.9.1.1(c).

[5] https://wiki.jenkins.io/display/JENKINS/Logging
[6] https://support.sonatype.com/hc/en-us/articles/213464768-Nexus-2-Logging-Guide
[7] https://docs.gitlab.com/ee/administration/logs.html

A.12.1.3(a)

Jenkins can be installed with vSphere plugin[8]. It should be enforced that systems have to be created, respectively managed via that plugin (via pipeline DSL). Furthermore, it has to be enforced that the vSphere has monitoring enabled which can be done via the *Application Programming Interface (API)*[9].

A.12.1.4(a)

Jenkins preserves the console log (log of all execute commands in a build) which might serve as an audit log.

A.12.4.1

See A.9.1.1(c).

A.12.4.2(b), A.12.4.3(a)

All delivery systems offer no means to edit logs. The systems should have correctly configured permissions, respectively access rights such that no unprivileged user can access the logs. As for privileged users, this is more difficult: The systems should have personalized log-in data which at least helps regarding traceability of manipulations. However, there has to be a root user for some operations and given that a person can physically access systems, manipulation of logs can never be ruled out. An option is to employ regular backups, respectively snapshot them, such that changes are observable.

A.12.4.3(b)

See A.9.1.1(c).

A.14.2.4

A minimal and imprecise solution is to search the artefact id of every artefact which is published to an internal repository (in the Nexus) in a public repository (e.g. the maven central repository). If a match is found there is an increased likelihood that the internal artefact is a modified external artefact.

[8] `https://wiki.jenkins.io/display/JENKINS/vSphere+Cloud+Plugin`
[9] `http://vmware.github.io/vsphere-automation-sdk-rest/6.5/operations/com/vmware/appliance/monitoring.list-operation.html`

A.14.2.7

The Nexus should proxy external repositories, thus only the proxy repositories have to be monitored. Monitoring here includes summarized reports of proxied artefacts, respectively new artefacts.

A.16.1.2

See 16.1.4, where we covered not only the assessment, but also the reporting.

A.16.1.7

See A.9.1.1(c).

A.18.1.3

Regularly snapshot and backup data, respectively log directories.

A.18.1.4

At least long-term archived logs shall be scanned on all user names existent in LDAP and be replaced with the associated user ids.

6.7.4 3. Change Management

A.12.1.2(a)

Jenkins stops build on errors in any build step (except explicitly configured otherwise) and does not perform any build step after the failing one.

A.12.1.2(b)

Forcing of two-man principle can be realized by protecting the master branch (and the release branch) of every project. This means that one can not commit (respectively push) directly to those branches. Gitlab merge-request approvals might help too[10]. Regarding Jenkins, one can use the multi-branch plugin[11] to automatically create a job per branch. Via pipeline DSL it is possible to e.g. enforce a release job.

[10] https://docs.gitlab.com/ee/user/project/merge_requests/merge_request_approvals.html

[11] https://wiki.jenkins.io/display/JENKINS/Pipeline+Multibranch+Plugin

A.12.1.2(c), A.12.5.1(a)

Use IaC, configuration management and a pipeline DSL. The pipeline management system should not offer the possibility to manually create pipelines / jobs. For Jenkins we can restrict the ability to create jobs with the *Role Strategy* plugin[12].

Furthermore, every project within the pipeline management system has to have a repository of a version control system as source and has to be created from such a repository. We can force this by only allowing the Gitlab as source code management.

Jenkins should be only allowed to be deployed via a pipeline (for which organization again has to enforce versioning).

A.12.1.2(d), A.14.2.3

Jenkins out-of-the-box (or with plugins) supports triggers, among others comprising version control triggers[13], scheduled triggers, etc.. Triggering on operating platform changes is more difficult. One approach might be to use a file system trigger plugin[14] in combination with writing a specific file in regular intervals on the systems. For Windows, writing such a file could be done with *Get-WinEvent -LogName Setup | where{$_.message -match "success"} | select -First 1.* For Linux one could let the FSTrigger plugin *monitor* the dpkg.log..

A.12.5.1(b)

First, release jobs have to keep full build history which can be forced in pipeline DSL. Second, since releases are pushed to some kind of artefact storage the log should contain the exact release information (at least for maven). If that is not the case, we should explicitly log the release information which can be enforced in the pipeline DSL.

A.14.2.2(a)

See A.12.1.2(b).

A.14.2.2(b)

In Gitlab it is possible to set push-rules[15].

[12] https://wiki.jenkins.io/display/JENKINS/Role+Strategy+Plugin
[13] https://plugins.jenkins.io/gitlab-plugin
[14] https://wiki.jenkins.io/display/JENKINS/FSTrigger+Plugin
[15] https://docs.gitlab.com/ee/push_rules/push_rules.html

6.7.5 4. Artefacts and Systems (ONLY internal aspects)

A.10.1.1, A.10.1.2

Use available static code analysis tools as mandatory build steps[16]. Since we reference this aspect in many other aspect, we want to mention here this list of security related static code analysis tools[17].

A.12.1.1

One could integrate a repository linter[18] as a mandatory build step to check for e.g. the existence of a *README.md*.

A.12.1.4(b)

Force VM instantiation of templates to different clusters.

A.12.1.4(c)

Deployment to operational systems should only be allowed if build, tests and deployments to testing, respectively development systems was successful.

A.12.1.4(d)

see A.12.1.4(c).

A.12.2.1(b)

See A.10.1.1, A.10.1.2, e.g. use vmray[19] to scan jars for malware.

A.12.2.1(c)

See A.12.1.2(c), A.12.5.1(a); enforce the use of the vSphere plugin for newly created systems (based on pre-made templates). Those templates shall include a virus scanner .

[16] like `https://cryptosense.com/support/static-crypto-scanner/`
[17] `https://www.owasp.org/index.php/Static_Code_Analysis`
[18] like `https://github.com/todogroup/repolinter`
[19] `https://www.vmray.com/cyber-security-blog/malware-uses-java-archive-jar/`

A.12.5.1(c)

Do not allow any installations on the created systems in A.12.1.2(c), A.12.5.1(a) (and also A.12.2.1(c)), respectively only allow changes to systems (templates) via IaC.

A.12.6.1

Integrate dependency-check, NPM audit, etc. as mandatory build steps into the process. Eventually use tools like dependency-track[20] for advanced features.

A.13.2.3(b)

Force the installation of a firewall (see A.12.1.2(c), A.12.5.1(a)) and e.g. close port 80.

A.14.1.1

see A.6.1.5.

A.14.1.2

see A.13.2.3(b).

A.14.1.3

see A.14.1.2.

A.14.2.1(a)

see A.10.1.1; use any static code analysis tool.

A.14.2.5

see A.10.1.1, A.10.1.2, A.14.2.1(a), etc..

A.14.2.8

see A.10.1.1, A.10.1.2.

[20] `https://dependencytrack.org/`

A.14.2.9

Use e.g. Cobertura[21] as a mandatory build step, respectively gateway.

A.18.1.2

see A.10.1.1, A.10.1.2.

6.7.6 5. Other Policies/Requirements

A.6.1.5

Use and enforce some kind of pipeline DSL to achieve mandatory process *steps*.

A.12.1.3(b)

Nexus, Gitlab and Jenkins all both offer an API and extensibility via plugins. With those means, obsolete data should be cleaned.

A.12.3.1

Use IaC and configuration management which certainly helps regarding the ease of a backup. Other than that, this requirement is rather fuzzy and we do not have any influence on the management of data which is specific to each application.

A.12.4.4

Force same (possibly local) *Network Time Protocol (NTP)* server on all systems via IaC.

A.13.2.1

Nexus, Gitlab and Jenkins all support TLS.

A.14.2.1(b)

We have no influence on Gitlab, but it e.g. offers TLS (see A.13.2.1).

[21] `https://cobertura.github.io/cobertura/`

A.16.1.4

Gitlab offers in the paid plan audit events[22]. Other than that there exists the */events* api endpoint which returns repository events. Both resources could be used by an additional application to create alerts based on certain metrics.

For Jenkins there are various possibilities, e.g. the Audit Trail Plugin[23] or the Prometheus metrics plugin[24]. Prometheus then can send out alerts.

Nexus also has a logging endpoint which could be post-processed by an additional application to create alerts as well as an auditing functionality[25].

A.17.2.1

All delivery sytems can be run as *High Availability (HA)*[26] [27] [28]. However, availability requirements have to be checked before anyway.

[22] https://docs.gitlab.com/ee/administration/audit_events.html
[23] https://wiki.jenkins.io/display/JENKINS/Audit+Trail+Plugin
[24] https://plugins.jenkins.io/prometheus
[25] https://help.sonatype.com/repomanager3/configuration/auditing
[26] https://help.sonatype.com/repomanager3/high-availability
[27] https://about.gitlab.com/solutions/high-availability/
[28] https://jenkins.io/doc/book/architecting-for-scale/

7 Discussion

> We spend our time searching for security and hate it when we get it.
>
> John Steinbeck

Contents

In the process of creating the SoA, we already regularly consulted and evaluated the state of the SoA with the company's security expert who is responsible for the ISO certification. Thereby, we mainly focused on the applicability and our understanding of the ISO 27001. Nonetheless, we also want to evaluate the applicability from another point of view, namely the DevOps team's view. In the following two sections we evaluate two different aspects. The first one is the already mentioned evaluation of the SoA. Thereby, we deal with the goals, the setup, our expectations and finally the results of this evaluation. In the second one, we then evaluate our maturity model, respectively DevOpsSec model and deal with threats to validity of our whole thesis.

7.1 SoA

7.1.1 Goals

We have three goals for our evaluation. They can be found in the following. The sub-bullets describe the concrete matter of evaluation:

1. Validation of SoA
    - Assess applicability in general

- Asses (degree of) correspondence of functional requirement to control

2. Applicability to CD of our industry partner
   - Assess applicability to CD of our industry partner of the (theoretically) applicable controls, respectively functional requirements
3. State / Degree of realization of controls at our industry partner
   - Get information regarding the state of realization (realized, planned, etc.)

7.1.2 Setup

The evaluation setup is as follows. We choose 2 team members of the DevOps team for evaluation. We set up a workshop, respectively a meeting in which we deal with every single ISO requirement one after another. The criteria according to which every requirement is being assessed can be found in section 7.1.1 and the respective scales in section 7.1.2. Matter of evaluation is section 6.1.

Scales

We want to obtain quantitative findings on the following (already stated) aspects for which we also state the scales:

- Applicability in general: {Unknown, Not applicable, Applicable} (nominal)
- Degree of correspondence of functional requirement to control: [-1,3] (interval; 3 for *Full Correspondence*, 0 for *No Correspondence*, -1 for *Unknown*)
- Applicability to CD of our industry partner: {Unknown, Not applicable, Applicable} (nominal)
- State of realization at our industry partner: {Unknown, Not planned, Partly Planned, Planned, Partly Realized, Realized} (ordinal)

7.1.3 Results

First, we validated all 88 controls of our SoA. Thereof, we considered 50 to be applicable before the evaluation and 38 not applicable. Important to note is that due to the limited time of this evaluation and the large extent of the SoA, we only considered the controls without distinguishing between internal and external. Figure 7.1 shows our validation resulted in eight controls being additionally considered applicable.

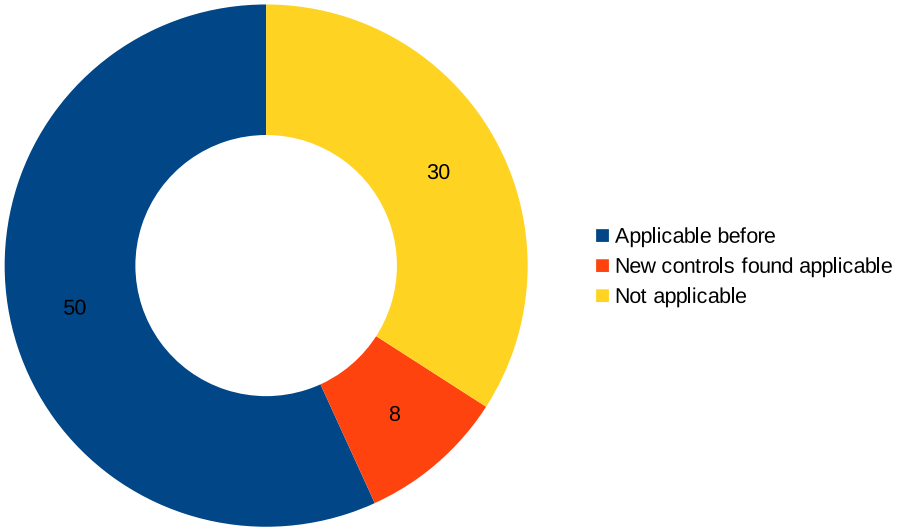

Figure 7.1: Number of applicable controls including new ones

For our industry partner, we evaluated all aspects applicable, respectively not applicable which we also considered applicable, respectively not applicable before the evaluation. We did not evaluate the applicability of the eight newly considered applicable controls, however, we consider it likely that they are also applicable to our industry partner since of the 50 applicable controls in general, all of them were also applicable to our industry partner.

The eight controls additionally considered applicable after validation are (including the reasoning):

- A.6.2.1: Use two factor authentication for CD systems
- A.8.1.1, 8.1.2: CD system already holds asset information (Gitlab, Nexus are nothing else than asset collections)
- A.8.1.3: Might also include user permission management (access to assets)
- A.8.2.2: Originally considered as a more abstract responsibility (e.g. label Jenkins assets once), however, the DevOps team e.g. already labels artefacts according to their availability (build artefact vs *Release To Manufacturing (RTM)* artefact)
- A.8.2.3: Restrict access to respectively labelled artefacts (see A.8.2.2)
- A.13.1.1, A.13.1.3: For VMs or Docker containers the DevOps team manages networks

Second, we validated the degree of correspondence between our functional requirements and the controls of the ISO 27001. Figure 7.2 shows the

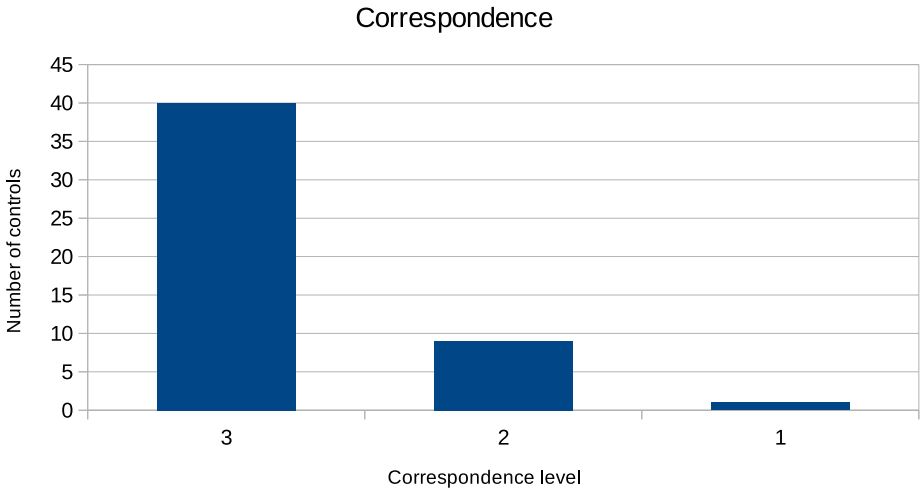

Figure 7.2: Overview of assessment of correspondence

distribution of the values (from one to three), the most frequent value being the *three*.

Finally, we evaluated the state of realization of the controls of our industry partner. An overview is shown in Figure 7.3.

We see that of the 50 applicable controls to our industry partner, around two thirds are either partly realized or realized where both states are pretty similar with 19 to 17 controls. Seven controls were not yet considered regarding planning at all. Two are partly planned and five are planned in form of cases.

Furthermore, we discussed many more aspects in detail. In the following we highlight some of the more interesting results of those discussions thereby including many of the controls with a lower evaluated correspondence. For that, we focus on those aspects which are of general interest and no specific aspects of our industry partner:

- A.9.2.1: Not all systems support LDAP, e.g. some require windows domain
- A.9.2.2: Even the applications with LDAP integration do not always (immediately) withdraw user access rights if the LDAP account is disabled
- A.9.2.5: If using Gitlab as authentication provider only send project, respectively namespace owners of Gitlab notifications in order to reduce spam
- A.9.4.4: Restriction of outgoing connections might help, respectively implement control

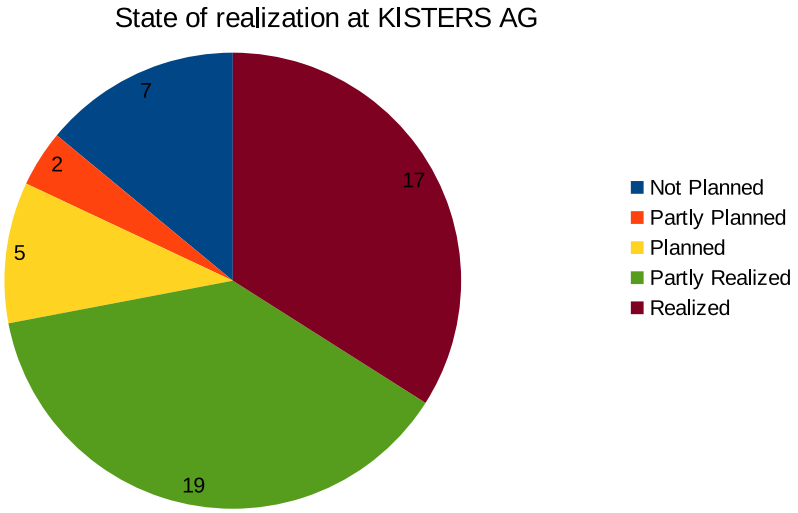

Figure 7.3: Current state of realization at our industry partner

- A.12.1.2: Change of permissions shall also be considered
- A.14.1.3: Also relevant and important for delivery systems itself, considering e.g. only releasing half of the artefacts due to connection problems

7.1.4 Discussion

After all, we see basically two interesting aspects. The first one is the eight aspects which were newly found to be applicable. This is the point where we have come full circle from our challenges. The problem regarding the ISO 27001 is its fuzziness. There are many aspects where CD can support the ISO 27001 in some way. This shows e.g. for control A.8.2.2 which we considered not to be the responsibility of the delivery systems, but more an organizational responsibility to label the delivery systems in general. Practice shows that this labelling is not only useful in such an organizational way, but also for concrete artefacts which means that we should definitely comply to the ISO 27001 here with CD.

The second aspect deals with the state of realization at our industry partner. We see that only about a third of the controls is already realized. However, also about a third is partly realized. And this is again the point where we have come full circle from our challenges, here, the intricacy of the ISO 27001 controls and therefore also of the extracted functional requirements. Many requirements are so broad that it is difficult to fully realize them after all.

Finally, one of the evaluation results of the SoA is a high degree of correspon-

dence between the controls and our functional requirements. Even though this is a good result regarding our thesis, we still consider some aspects being to broad and think that the SoA can be further refined in order to get more detailed requirements such that a company can fully implement all applicable controls with CD.

7.2 Discussion of Assignment Statistics

In section 6.1.3 we observed that there are only three requirements assigned to model in total. This can be explained by the vast amount of information the delivery systems carry, respectively contain. Thus, conceptually nearly all the information needed is somewhere available. This is very interesting to see since it shows that we can fully automate nearly all the aspects we found to be applicable.

However, in a technical realization there might be more requirements which require user input, because not every information might be technically easy to obtain.

Another aspect which we want to discuss here are the categories Tony Hsu considered to be in the responsibility of DevOps in contrast to our detailed assessment on applicability. In accordance with Tony Hsu, we can see in section 6.1.3 that we did not found any control to be applicable in sections 5, 7, 8, 11, 15 of the SoA. However in section six we found one out of seven controls to be applicable, namely the quite general control that information security shall be addressed in project management. After all the areas of responsibility as stated by Tony Hsu (see Figure 2.4) are sensible.

7.3 Discussion of Models

In this section, we discuss the dependency, the maturity and finally the mapped model.

Figure 6.1 only has five inter-category dependencies. This indicates that our choice of categories is sensible, since most dependencies are intra-category dependencies.

Interesting to see in Figure 6.2 is the distribution of the number of requirements over the different levels. The highest maturity level, the *Artefacts-oriented* level, comprises the highest number of requirements. Furthermore, the majority of its requirements are *process* related requirements. The number of requirements decreases in both the middle level, the *System-dependent* level, and the lowest level, the *System-oriented* level. In contrast to the high number of *process* related requirements in the *Artefacts-oriented* level, the

middle *System-dependent* level includes most *system* related requirements. After all, this appears to be sensible since the process has to be supported by the system. Finally the lowest level of maturity is characterized by containing rather general requirements mostly related to the *system*.

All in all, this means in order to get a high degree of ISO 27001 compliance for artefacts and systems from a CD viewpoint, we have to obtain a high level of maturity. This especially requires a high degree of ISO 27001 compliance of the *system* itself first.

Regarding our mapped model (see Figure 6.3), we stated that on average an intermediate level of maturity is required for most security requirements. Due to the intermediate level being a level where full automation in many aspects is required, it is a sensible result that this is the level which is on average required by our security requirements, because our goal with CD is to automate as much as possible. Furthermore, we discovered some hotspots. This coincides also with our realization, where some technical realizations like using IaC were important for many requirements and this also coincides with the realizations at our industry partner where the same fact applies.

8 Conclusion

Final thoughts are so, you know, final. Let's call them closing words.

Craig Armstrong

Contents

In this thesis we examined the ISO 27001 and its applicability to CD, respectively its implementability with CD. The goal was to provide guidance and models in order to help organization's CD to maximally support, comply to and implement the ISO 27001 and in general to increase security when using CD. We were interested in the state of realization at our industry partner, as well as the possible degree of automation of the ISO 27001 with CD.

Therefore, we created the SoA in which we assessed the applicability, the implementability with regard to *model*, *process*, or *system* and finally formulated functional requirements. Thereby, we found the ISO 27001 to be highly automatable. We then categorized those functional requirements in order to structure them, especially for the further usage in our models. In the next step, we created the first model, namely the realization roadmap in which we assessed the requirements according to their estimated required effort. After that, we built the dependency model which we then used to create the maturity model. Thereby, the maturity model showed that in order to get a high degree of ISO 27001 compliance for artefacts from a CD viewpoint, we have to obtain a high level of maturity. This in turn requires a high degree of ISO 27001 compliance of the *system* itself first.

Lastly, we mapped our requirements to the maturity model by Rehn et al. which yielded the result that for our security requirements on average an intermediate level of CD is required.

Finally, we discussed the SoA itself and the state of realization at our industry partner with the result that it is difficult to fully implement applicable controls of the ISO 27001. However, at our industry partner around two thirds of the controls are either realized or at least partly realized which we consider a good result.

8.1 Threats to Validity

Due to the challenges mentioned which already occurred to us in the discussion, there are two threats to validity which we would like to mention.

The process of transforming the ISO 27001 into a (multiple) model(s) includes many abstractions. We performed them with greatest prudence, but we can not guarantee that no information or control got altered in its understanding during this process or even that every control was understood correctly and as intended in the beginning. This is partly due to the fuzziness of the ISO 27001 (see section 3.1.2).

Furthermore, we had to map those controls to CD. We had to come up with concepts how ISO 27001 controls can be realized in CD. For some controls, there might be multiple solutions and for some other controls there are solutions where we did not consider any solution at all.

8.2 Future Work

In a next step one could technically realize every requirement for all delivery systems. As such, ISO 27001 compliance could be reached with the highest degree possible for CD by just using this system which performs in a (nearly) fully automated way.

Before, one should probably further refine the SoA. This could be done e.g. by comparing the results of a real assessment with the ones we obtained which was not possible in this thesis due to the timing of the ISO 27001 certification.

Additionally, one could create a metric such that one could calculate the degree of compliance to the ISO 27001 based on fulfilled requirements. Finally, every build could automatically include an audit report regarding the compliance to the ISO 27001 based on the individual process it passed.

Bibliography

[All] C. S. Alliance. *Cloud Control Matrix*. URL: https://cloudsecurityalliance.org/ (visited on 09/10/2019) (cited on pages 13, 14).

[BRP13] P. Boström, A. Rehn, and T. Palmborg. *The Continuous Delivery Maturity Model*. 2013. URL: https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model/ (cited on pages 5, 6, 73).

[Bsia] *BSI-Standard 200-1 - Information Security Management Systems (ISMS)*. Standard. Bundesamt für Sicherheit in der Informationstechnik, Oct. 2017 (cited on pages 3, 12).

[Bsib] *BSI-Standard 200-2 - IT-Grundschutz Methodology*. Standard. Bundesamt für Sicherheit in der Informationstechnik, Oct. 2017 (cited on page 21).

[BWZ15] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. SEI Series in Software Engineering. New York: Addison-Wesley, 2015. ISBN: 978-0-13-404984-7. URL: http://my.safaribooksonline.com/9780134049847 (cited on page 8).

[Che15] L. Chen. "Continuous Delivery: Huge Benefits, but Challenges Too". In: *IEEE Software* 32.2 (2015), pp. 50–54. ISSN: 0740-7459. DOI: 10.1109/MS.2015.27 (cited on page 4).

[Com+90] I. S. C. Committee et al. "IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos". In: *CA: IEEE Computer Society* 169 (1990) (cited on page 25).

[Fit+13] B. Fitzgerald et al. "Scaling agile methods to regulated environments: An industry case study". In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 863–872 (cited on pages 24, 25).

[FS14] B. Fitzgerald and K.-J. Stol. "Continuous Software Engineering and Beyond: Trends and Challenges". In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. RCoSE 2014. Hyderabad, India: ACM, 2014, pp. 1–9. ISBN: 978-1-4503-2856-2. DOI: 10.1145/2593812.2593813.

URL: http://doi.acm.org/10.1145/2593812.2593813 (cited on page 22).

[Gdp] *Official Journal of the European Union*. Legislation. European Union, 2016 (cited on page 12).

[Git] Gitlab. *GitLab Documentation*. URL: https://docs.gitlab.com/ (visited on 09/30/2019) (cited on page 74).

[HF10] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010 (cited on pages 4, 5, 7, 8).

[HL06] M. Howard and S. Lipner. *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006. ISBN: 0735622140 (cited on page 21).

[Hsu18] T. Hsu. *Hands-on security in DevOps : ensure continuous security, deployment, and delivery with DevSecOps*. Birmingham, UK: Packt Publishing, 2018. ISBN: 978-1788995504 (cited on pages 9–11, 18, 20, 25).

[Isoa] *Information technology — Security techniques — Code of practice for information security controls*. Standard. International Organization for Standardization, Oct. 2013 (cited on pages 20, 21, 39, 40, 42, 52).

[Isob] *Information technology — Security techniques — Information security management systems — Requirements*. Standard. International Organization for Standardization, Oct. 2013 (cited on pages 11, 33).

[Jen] Jenkins. *Jenkins*. URL: https://jenkins.io/ (visited on 09/30/2019) (cited on page 74).

[MCP17] H. Myrbakken and R. Colomo-Palacios. "DevSecOps: A Multivocal Literature Review". In: *Software Process Improvement and Capability Determination*. Ed. by A. Mas et al. Cham: Springer International Publishing, 2017, pp. 17–29. ISBN: 978-3-319-67383-7 (cited on pages 1, 17, 18).

[MO16] V. Mohan and L. B. Othmane. "SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps". In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*. 2016, pp. 542–547. DOI: 10.1109/ARES.2016.92 (cited on page 1).

[MR11] M. Merkow and L. Raghavan. "An ecosystem for continuously secure application software". In: *CrossTalk, March/April* (2011) (cited on page 24).

[OWA] OWASP. *OWASP SAMM Project*. URL: https://www.owasp.org/index.php/OWASP_SAMM_Project (visited on 09/16/2019) (cited on page 74).

[Sam] *Software Assurance Maturity Model - A guide to building security into software development*. Standard. OWASP, Mar. 2009 (cited on pages 22, 23).

[SLD18] A. Steffens, H. Lichter, and J. S. Döring. "Designing a Next-generation Continuous Software Delivery System: Concepts and Architecture". In: *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*. RCoSE '18. Gothenburg, Sweden: ACM, 2018, pp. 1–7. ISBN: 978-1-4503-5745-6. DOI: 10.1145/3194760.3194768. URL: http://doi.acm.org/10.1145/3194760.3194768 (cited on pages 8, 9).

[Son] Sonatype. *Repository Manager 3*. URL: https://help.sonatype.com/repomanager3 (visited on 09/30/2019) (cited on page 74).

Glossary

API Application Programming Interface

CAPEC Common Attack Pattern Enumeration and Classification
CCM Cloud Controls Matrix
CD Continuous Delivery
CSA Cloud Security Alliance
CVE Common Vulnerabilities and Exposures

DSL Domain Specific Language

EU European Union

GDPR General Data Protection Regulation

HA High Availability

IaC Infrastructure as Code
IP Internet Protocol
ISMS Information Security Management System

LDAP Lightweight Directory Access Protocol

NIST National Institute of Standards and Technology
NTP Network Time Protocol

OS Operating System
OWASP Open Web Application Security Project

RTM Release To Manufacturing

SAMM Software Assurance Maturity Model
SDL Security Development Lifecycle
SoA Statement of Applicability

TLS Transport Layer Security

URL Uniform Resource Locator

VM Virtual Machine