NOREA
DE BEROEPSORGANISATIE VAN IT-AUDITORS

DevOps in control

A study report by NOREA

Author: S. Gangaram Panday MSc RE CISA - Brightlyn

Update 2026: Workgroup DevOps & Agile in Control

Postbus 242, 2130AE Hoofddorp
phone: 088-4960380
e-mail: norea@norea.nl
www.norea.nl

Accountability

This study report has been published by the NOREA, the professional organization of IT auditors in the Netherlands, and has been developed to give Dutch qualified IT auditors (Register IT auditors, RE's) guidance in assessing the quality of Agile and DevOps practices.

Version control

Version	Date	Changes
1.0	09-2019	Publication of initial version by S. Gangaram Panday
2.0	01-2026	Updates to several paragraphs of this study report Updates to the control descriptions Merging of control 7 & 8 Addition of control 14 as a new control Addition of maturity levels

Participants of the DevOps & Agile in Control work group

- Sandeep Gangaram Panday MSc RE CISA (chair)
- Pieter Jolen MSc RE
- ir. Jean-Jacques Bistervels RE CIA CFSA CDPSE CRISC CCP
- Zubair Yaseen MSc RE RA CIA
- Than Son Nguijen
- Edwin Galama RE RA
- Paul van Kemenade
- Boris Cuijpers

The DevOps Control Framework is also available in Excel:
https://www.norea.nl/organisatie/kennis-en-werkgroepen/kennisgroep-software-development

Contents

1. Introduction

In recent years, Agile, DevOps and now DevSecOps have become the dominant approaches for modern software delivery, not only within technology-driven companies but also in highly regulated sectors such as financial services, government, and critical infrastructure. Organizations increasingly rely on autonomous, cross-functional teams, cloud-native platforms, automated pipelines, and integrated security tooling. Delivery cycles have accelerated dramatically, with many teams deploying multiple times per day using standardized CI/CD capabilities. Documentation, quality checks, and even security controls are embedded directly into tools, logs, and automated workflows rather than produced as traditional artefacts.

For IT auditors, risk professionals and security specialists, this shift introduces new challenges and expectations. Many established audit frameworks were designed for phased development, long release cycles, and manual approval processes. These assumptions no longer align with environments where changes are deployed continuously, infrastructure is defined as code, testing is automated, and system-generated evidence replaces traditional documentation. At the same time, regulators increasingly expect organizations to demonstrate operational resilience, secure software development, and robust change governance. Regulations such as DORA and NIS2 and updated ISO standards highlight the importance of understanding modern delivery practices and the risks inherent in high-velocity, highly automated environments.

This updated 2026 edition of the DevOps in Control study report aims to bridge the gap between modern engineering practices and the expectations of auditors and risk professionals. It provides practical guidance on assessing Agile and DevOps environments, supported by a control framework and maturity model that reflect the current state of the industry. The goal is to help auditors evaluate both the technical and cultural aspects of these practices, understand where automated controls can be relied upon, and recognize where additional assurance activities are needed.

1.1 Motivation and goal

NOREA is the Dutch association of IT auditors. As stated on its website the goal of NOREA is threefold:

1. Promote the quality of the professional practice of IT auditors
2. Promote the further development of the IT audit profession
3. Take care of the common interest of the members

The second goal, especially, is the reason NOREA identified the need to publish a study report on a control approach for new software development techniques that are being widely used nowadays. Several research papers and whitepapers conclude that DevOps indeed requires a different control approach. One important example is the published COBIT 2019 framework which mentions that DevOps “definitely requires specific guidance” [40].

DevOps is a fact and the number of organizations adopting the DevOps principles is growing rapidly, as appears from the availability of several detailed step-by-step implementation guidance with the inclusion of use cases of many organizations [1] [2]. Rejecting the transition to DevOps is not an option anymore. Especially because we also see application of these new approaches in (highly) regulated markets which is often the focus of many auditors. A specific example is the cloud.gov platform [4] of the US Federal Government, a Platform as a Service (PaaS) solution for US government agencies. This platform allows the use of Agile and DevOps methodologies, while at the same time meeting the requirements of a highly regulated environment (FedRAMP and FISMA) [5]. We want to take this as an inspirational example for applying these principles within highly regulated environments.

Or as stated by Gartner: "Every business is a digital business. Every company is a software company. The key to gaining and sustaining competitive advantage in digital business, and a role in a digital society, will be in the development and continuous improvement of new IT-enabled capabilities and services for customers" [41].

The goal for this study report is to provide IT auditors, but also other information security and risk professionals, with a basic introduction and a control framework to mitigate the key IT risks associated with Agile and DevOps principles and to evaluate the level of maturity of controls.

We have not specifically referenced which controls are required at a minimum for the Annual accounts audit because we want to emphasize that there is not one universal Agile or DevOps approach (as also emphasized in COBIT 2019). Each implementation, that we as auditors might observe, will have its own unique approach towards implementing the core Agile and DevOps principles. For example, a DevOps setup that is fully focused on the e-commerce front-end will be different from the back-end setup of a bank. In the end the IT auditor will therefore need to properly assess the specific implementation that needs to be audited and cautiously select the proper controls from the control framework presented in this study report.

1.2 Method of research and approach

The DevOps (maturity) control framework that is presented in this study report is built upon the ever-increasing number of articles, (research) papers, books and best practice models about Agile and DevOps (see the Reference list in Appendix A). The leading paper with guidance on DevOps auditing is the DevOps Audit Defense Toolkit which is currently the most elaborative control framework for DevOps and gives detailed work instructions on how to implement and assess change management within DevOps environments [6]. However, we stress that we wanted only to provide a 'lean and mean' framework which covers the most detrimental risks in a DevOps environment.

We additionally want to state that the vision, knowledge and approaches presented in this paper are also based on the authors experiences with auditing software development projects where Waterfall, Agile and DevOps techniques were applied. Additionally, several interviews with engineers, IT managers and project managers were part of the research.

Initially with the first version of the report, the control framework was not mapped with best practice IT governance or control models, because there was no fit with the models available at the time. At the end of 2018, ISACA published its 2019 upgrade of COBIT (Control Objectives for Information and Related Technologies), which acknowledges the need for a different control approach to accommodate auditing of DevOps environments. Because the COBIT frameworks are widely accepted and used by IT auditors below is a summary of the

most important changes and/or additions in COBIT 2019 compared to the previous version (COBIT 5) regarding this subject:

- Emphasis on the importance of tailoring the IT governance and control frameworks to the specific organizational context instead of using off-the-shelf control frameworks.
- Acknowledging the importance of the cultural aspect by mentioning that senior management must actively steer on achieving a different mindset and culture for delivering value from IT.
- Rectification of the often encountered (i.e., narrow) interpretation suggested by the GRC (Governance, Risk and Compliance) acronym. The GRC acronym itself implicitly suggests that compliance and risk represent the spectrum of governance ("we make the mistake that risk and compliance direct governance whereas they go hand-in-hand and support each other").
- Mentioning of open and flexible architectures and control frameworks, aligned to major standards, as one of the three principles of a governance framework.
- Adding the Design Factors and Focus Areas to the scoping and creation of the framework. DevOps is specifically included as one of the Focus Areas (because "DevOps is a current theme in the marketplace and definitely requires specific guidance").
- Based on the COBIT 2019 Design Guide, specifically the following COBIT controls are recommended to be included (and tailored) for DevOps environments: BAI02 (Managed requirements definition), BAI03 (Managed solutions identification and build) and BAI06 (Managed IT changes), along with a reference to a yet to be published DevOps paper. These controls have been included in our DevOps control framework in paragraph 3.4. The reader will therefore see that the presented control framework has been aligned with COBIT 2019 on several aspects.

Furthermore, alignment has been sought with the security control framework developed by the Secure Software Alliance (SSA) specifically for Agile software development. This framework provides security related controls for all phases of software development and was initially created by a group of Dutch software security firms supported by the Dutch Ministry of Economic Affairs. The SSA framework is free for use and can be downloaded from their site [15]. The framework consists of 4 control domains: Context, Threats, Implementation and Verification. In this guide we have mostly focused on the controls within the Implementation and Verification phases of the SSA framework.

1.3 Limitations on the scope

We developed this study report using an Agile approach as well, which means that this study report is the second iteration. Based on market demand and interest, we will define the focus point for our next iteration. With this approach we also want to emphasize that this second iteration is by no means meant to be complete and covering the full picture but has a focus on the essential risks and associated controls that we gathered based on feedback from the first iteration.

1.4 Layout of the report

In chapter 2 we briefly introduce the Waterfall methodology as an example of a more traditional software development methodology because Waterfall is the most well-know methodology among IT auditors. Chapter 2 also includes an introduction of the Agile and DevOps software development principles to provide a comparison between these 3 approaches and to be able to better understand the control framework presented in paragraph 3.4.

In chapter 3 we present the approach on auditing Agile and DevOps environments. In paragraph 3.1 we introduce the auditor with some guidance to make sure the right type of controls is selected. In paragraph 3.2 a short introduction of the relevance and impact of the Agile and DevOps culture is given. The reader is provided with some references of models that can be used to measure the Agile and DevOps culture within a team which are useful tools for the auditors. In paragraph 3.3 guidance is provided on which testing approach to select and finally in paragraph 3.4 the Agile and DevOps (maturity) control framework is presented.

In chapter 4 the updated conclusion is presented.

2. Waterfall, Agile and DevOps

2.1 Waterfall

Wikipedia provides the following definition (July 2019): “The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design. In software development, it tends to be among the less iterative and flexible approaches, as progress flows in largely one direction ("downwards" like a waterfall) through the phases of conception, initiation, analysis, design, construction,
testing, deployment and maintenance.” [18].

To properly understand the waterfall software development methodology, it is important to take a closer look at the time when waterfall originated and how it originated. Below we attempt to give a short summary of this history:

- The basic structures of the waterfall model in software development as we currently know it were first introduced in 1956 by Herbert D. Benington [18], although not under the name of “waterfall”. It consisted of 9 phases, see appendix C for an overview [19].

- In an article published by Winston W. Royce in 1970 this model of Benington was for the first time more formally documented into a methodology [20]. At that time the term “waterfall” was not used either. Important to note is that Royce explains that the typical downward flow of the waterfall method is flawed. Royce introduces iteration to this model. However, still at a quite modest level: “as a step progresses and the design is further detailed, there is an iteration with the preceding and succeeding steps but rarely with the more remote steps in the sequence.”

- It appears that the first use of the term “waterfall” was in the 1976 paper by Bell and Thayer [21]. In this paper they refer to the top-down software development methodology of Royce and call this approach the “the waterfall of development activities” [21].

- In 1983 Herbert D. Benington, who initially introduced the waterfall concept, republished his article with a new foreword in which he explains that in his initial publication he did “omit a number of important approaches, which I will say a little more about below” [20]. Some of the additional approaches that he mentions are:

    - The application of a structured, highly disciplined engineering mindset for developers.

    - The waterfall top-down approach is not to be interpreted too literally: “This attitude can be terribly misleading and dangerous. To stretch an analogy slightly, it is like saying that we must specify the characteristics of a rocket engine before measuring the burning properties of liquid hydrogen” [20].

    - Experimental prototypes are important to develop and based on the result change the specifications.

- The biggest mistake his team made: the attempt to make a too large release. He would now focus on smaller changes and test and evolve the system from there.

Strangely enough, at that time the sequential / one-direction waterfall methodology was already embedded within the industry and his feedback did not result in changing the methodology (of which he was the founder).

Key characteristics of the waterfall methodology [23]:

- Run from a project organization (timely organization).
- Rigid planning.
- Sequential one-direction flow of activities.
- Different and separated teams per phase.
- Need for extensive documentation (because teams are working in separation).
- Hand-over to normal IT operations department after project is delivered.

In practice we see the waterfall methodology mostly being implemented with the assistance of the PRINCE2 project management framework.

Figure 1 provides an overview of the waterfall phases which form a software development sequence (a "waterfall"). In some articles these phases appear with slightly different names.

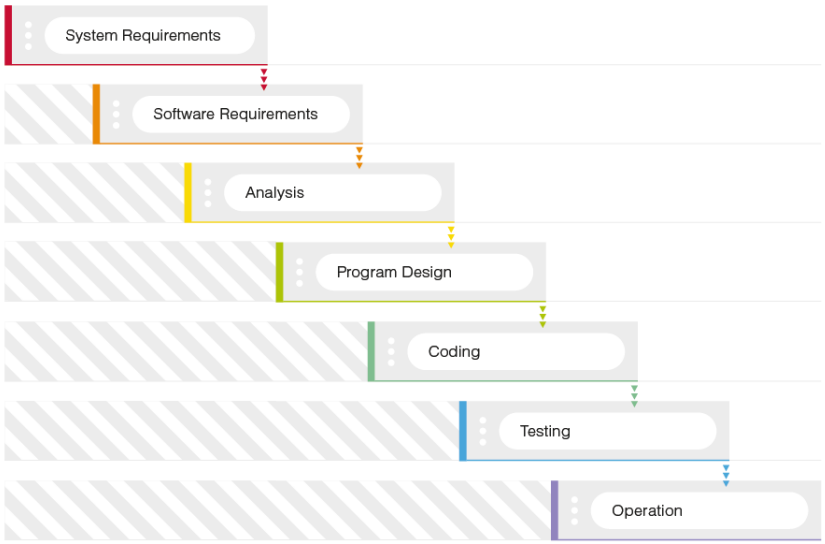

Figure 1: The waterfall model [23]

2.2 Agile

The Agile approach was introduced as the natural counterpart of the Waterfall methodology to resolve issues associated with the latter. Agile development does not apply a plan-driven approach but an iterative approach. It is not defined as a methodology but as a set of principles to be applied together in order to achieve an intended goal. The 12 Agile development principles and their origin can be found in the Agile Manifesto [10]. The Agile Manifesto was defined to enable better ways for developing valuable software more rapidly (principle 1).

In most literature Agile development is perceived as an evolution from several practices and alternative methodologies for software development designed in the 90's. Examples of such practices are the Theory of Constraints [7]. Lean Manufacturing [8] and, very relevant to auditors, the famous Deming's principles [16], which had already proven their effectiveness in the manufacturing and automotive industries [14].

Key agile development characteristics according to van Casteren [23]:

- Iterative development with frequent visible results as output (principle 3).
- Focus on interaction and communication (principle 6).
- Reduction of resource-intensive intermediate artifacts e.g. backlog vs formalized requirements document (principle 10).
- Feature planning and prioritization performed in short iterative cycles (principle 3).
- Fast decision making (principle 4).
- Close customer relationships for timely assessment and feedback on increments (principle 4).

Given the fact that the Agile development approach is not a methodology, there are currently many Agile development best practices available, which all share common characteristics but each having their own nuances and specializations. We can therefore consider "Agile" to constitute an umbrella term, which covers among others; Scrum, Kanban, Extreme Programming (XP), Crystal and Lean software development practices. In figure 2 below an overview is provided of Scrum for implementing agile principles.

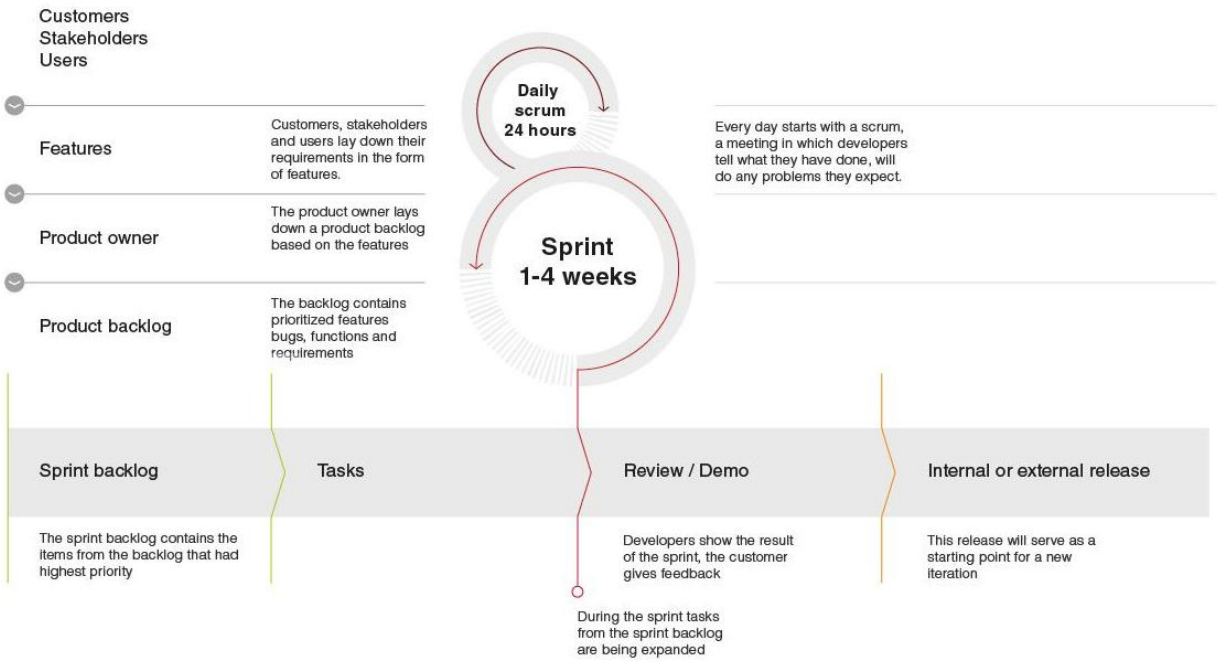

Figure 2 Scrum schematic overview [23]

We still see the main phases of Waterfall in the Agile approach; however, they appear in a shorter and iterative fashion. Basically, each iteration is a self-contained mini project with activities that relate to the Waterfall phases.

Tools & technology

In order to achieve the desired agility, the use of suitable development tools, a high level of automation and a constant drive for technical excellence is a pre-requisite (principle 9). Effort is put on the practices to remove the barriers in collaboration more effectively together with stakeholder's, while being able to welcome and respond to changes. The tools used for Agile development are typically limited to software development activities, the reason being that development and operations teams are still separated. Therefore, software development and software deployment/release (operations) are still managed by separate teams, using separate ways of working.

Out of this focus on automation rose the Infrastructure as Code practice and the widespread use of Version Control Systems combined with automated builds (Continuous Integration) and automated deployments (Continuous delivery). These practices are closely related to Agile and almost always applied by Agile development teams.

Version control

A Version Control System (VCS) allows developers to work on code from different workstations at different locations (pull) while still being able to integrate their code into a single repository (merge), which can be used later to deploy the entire system. The VCS is also used to document and track system configuration files (see Infrastructure as Code). The consistent use of a VCS is considered the first step on the path to Continuous Integration (CI) and Continuous Delivery (CD); see below.

Infrastructure as Code

Infrastructure as Code (IaC) is the practice to manage and provision infrastructure through code and automation instead of manually (e.g. by logging in with SSH into a host and executing commands). IaC is for example used to create and change containers, instances, servers, and complete environments from already created scripts/templates. By maintaining these automation scripts in the VCS, a fast repeatable and auditable method is achieved for the creation and maintenance of infrastructure components based on best practices (e.g. security baselines).

With IaC, several powerful software development practices have been adopted within the operations field such as use of VCS, peer review, automated testing, release tagging and release promotion [25]. The application of IaC also greatly enhanced the audit of the configuration management process for auditors.

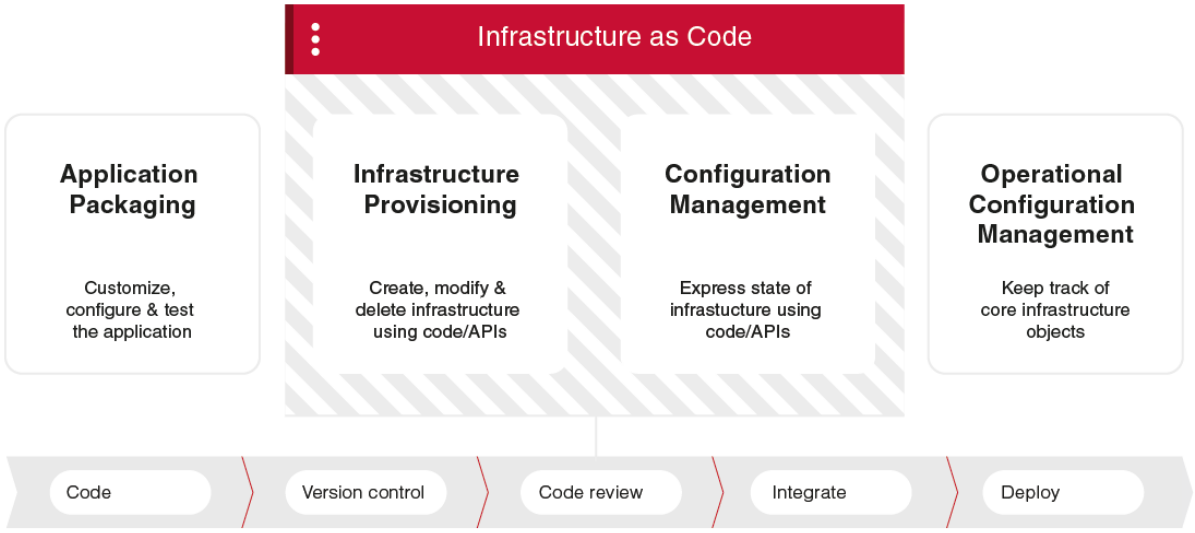

Figure 3 Infrastructure as code [38]

2.3 DevOps

Gradually Agile development expanded into other areas within IT of which primarily IT operations. This union of previously separated development and operations teams has been called DevOps and is basically the next step in the evolution of Agile to further increase rapid value delivery to the end customer by streamlining and automating the entire software delivery lifecycle. As such, DevOps is not a methodology nor an approach but a philosophy and a way or working to enable collaboration between previously separated teams/departments, using a high level of automation. To achieve this, several other methodologies are applied, some of which even outside the software development field (e.g. social psychological beliefs).The goal of DevOps is to reduce lead time in all the software delivery steps from months or weeks to minutes while maintaining control and consistency across all environments. This is only possible by applying a high degree of automation in the software delivery lifecycle which in DevOps terminology is called the Delivery pipeline. The focus of this automation lies in the integration of the end-to-end activities needed to transform a vision to a workable feature. Next to the high focus on automation of the complete pipeline, DevOps has an important prerequisite, namely the culture. It is explicitly emphasized that the cultivation of the right culture is critical to make previously separate teams working together. From all the years of applying Agile in software development, one important lesson learnt was that the human factor appeared to be a limiting factor in increasing the level of agility.

Taking the above into account, we have formulated the following definition for DevOps: DevOps is the union of, at least, software development and IT operations activities in an environment that has incorporated the accompanying cultural and technical principles to deliver business value at a high frequency.

If we look at the available literature, we see that the DevOps practice is summarized in three core principles which we see as supplementary principles to the Agile principles [1]:

1. Flow: the Delivery pipeline facilitating automated build, testing, integration and deployment, to enable fast flow from business to development to operations combined with an emphasis of small changes over big releases.

2. Feedback: functional monitoring to identify issues and communicate feedback fast to everyone involved. For example, Blue/Green deployment[1], A/B testing[2] and Canary releases[3].

3. Learning: the continual learning process from incidents (e.g. blameless post-mortems[4]) and failures (e.g. chaos monkey[5]) and of as much actual users of the system by connecting technical experts with these actual users.

Different categories of DevOps

Because DevOps is not a formally defined and documented methodology – unlike frameworks such as PRINCE2 -, there are many types of DevOps implementations, and almost every DevOps team represents a unique approach. Due to the diversity of DevOps implementations, understanding DevOps typologies offers useful context for identifying and assessing associated risks. The DevOps Topologies organization presented 8 DevOps anti-types and 9 DevOps collaboration types (ranging from an effectiveness level of low to high) to create awareness regarding the several different, most common 'flavors' of DevOps implementation [24]. For proper application of the control framework presented in paragraph 4.3, it is crucial to be aware of the model used by the team that is to be assessed.

DevOps practices

In Agile teams, automation (and subsequent improvement) of the software development process could already be enhanced by using VCS and IaC. With the application of DevOps practices, the automation evolved further into the concepts known as CI/CD which are shortly introduced below:

- Continuous Integration (CI): Martin Fowler introduces the following definition for CI: "a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early" [26]. CI is an enhancement built upon the use of VCS. CI is often one of the drivers of Agile practices.

- Continuous Delivery (CD): As an extension of CI and the next step in incremental software delivery, CD ensures that every version of the code in the CI repository that

---

[1] A deployment technique that requires two identical production environments (blue and green) which can be used to deploy a new release to e.g. blue to gain feedback on the working of the release and after successful feedback switch the router to so send all incoming requests to blue (instead of green). There are several nuances and different approaches available regarding the use of this technique.

[2] A/B testing is a way to compare two versions of a single variable, typically by testing a subject's response to variant A against variant B and determining which of the two variants is more effective.

[3] Canary release is a technique to reduce the risk of introducing a new software version in production by slowly rolling out the change to a small subset of users before rolling it out to the entire infrastructure and making it available to everybody [45].

[4] A postmortem is a written record of an incident, its impact, the actions taken to mitigate or resolve it, the root cause(s), and the follow-up actions to prevent the incident from recurring. It is done with the focus on identifying the contributing causes of the incident without indicting any individual or team for bad or inappropriate behavior (source: Google).

[5] A program that randomly chooses a server and disables it during its usual hours of activity.

has been tested can be released at any moment. This is often referred to the concept of “maintaining code in a deployable state”. It is achieved through a set of practices and methodologies designed to improve the process of software delivery and ensure reliable software releases. Leveraging automation, from CI builds, to (security) testing, to deployment, CD involves all dimensions of the development and operations organization. Ultimately, it enables the systematic, repeatable, and more frequent release of quality software to end customers [27].

- Continuous Deployment: As an extension to Continuous Delivery (CD), Continuous Deployment focusses on executing the deployment to production automatically after every change. It is the set of practices to enable frequently deploying small code changes to production by removing all manual steps in the Delivery pipeline. If a deployment causes a problem, it is quickly and reliably rolled back using an automated process. Through this robust automation, rollbacks are a reliable way to ensure stability for customers and at the same time are convenient for the developers because they can roll forward with a fix as soon as they have one.

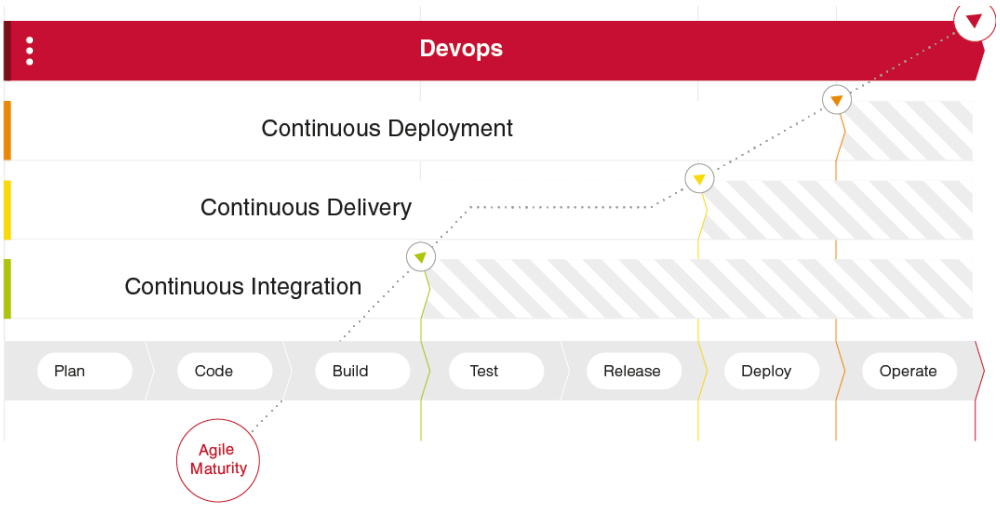

Figure 4: Comparison of CI /CD, Continuous Deployment

Tooling overview (Appendix D)

In the previous two paragraphs several technological principles have been explained. There is an actively growing number of tools becoming available to achieve this level of automation. Xebia labs has created a ‘Periodic table of DevOps tools’ [42] to provide an overview of the most used tools for each of the phases in the delivery pipeline (see Appendix D). This overview is important as it gives the auditor insights in the level of automation applied in the Integration and Delivery pipeline which impacts the audit approach to be applied.

Documentation

The logical result of the high-level of automation in both Agile and DevOps teams generates off course a lot of source code versions. This source code has become the new (audit) documentation. With all steps and activities registered in the VCS and logging of all code changes, including what changed and by whom and when, there is no or less need to create several of the traditional formal documentation. However, environment setup instructions and

diagrammatic representations of the architecture are useful for bringing other engineering team members up to speed on the system and sharing knowledge. The goal of the documentation has changed from being imperative to understand the environment to being informative for sharing purposes.

The Shared Services organization

Tooling and technology are important within DevOps to make the high level of automation possible. It is therefore often seen that teams have a lot of freedom in the selection, use and configuration of tools to gain experience on the best solutions. Research shows that when the DevOps teams and their practices start maturing, naturally the focus then shifts to normalization and standardization of the tools and services used within the organization [3]. This standardization is also fueled by the high degree of collaboration within the different (DevOps) teams within the organization. This increases the development of proven best practices. As a result, it is observed that gradually Shared Services teams are formed. These teams now take over the management of several of the tools and best practices used within the DevOps teams (for example the tools needed for the Integration and Delivery pipeline). Also, often DevOps teams make use of (public) cloud resources, the management of which now shifts towards the Shared Services team. This is confirmed in the State of DevOps report 2018 in which is stated that DevOps teams report an enhancement in their delivery quality and a further gain of efficiency by acquiring tools and services form Shared Services teams [3]. As such a software delivery 'ecosystem' is created where more teams are part of the management of an application's changes. The result is a longer 'software chain of custody'. In the next chapter the impact on the audit is further elaborated.

Test strategy

In order to understand whether new software will work in production, developers need to run tests on their software in production-like environments which are nearly identical to the production environment, as any deviation in the test environment compared to the production environment increases the chance of running into problems later in the pipeline. Following the principle of Agile to reduce the number of handoffs, it would be best if developers could create these production-like environments in a self-service manner. This is possible by using IaC practices.

Automating testing allows speeding up the test process significantly compared to manual testing and is less time-consuming and less dependent on the quality of individual testers. By automating tests, developers can run (some of) the tests directly after having finished a code change. Not only can tests therefore be performed earlier in the development process, it also reduces the number of handoffs between testers and developers and acts as a tollgate during propagation of releases from development to test to production(like) environments.

Some examples of tests to be executed, presented in the order of easiest to more difficult to automate are:

- Unit Tests: testing a single method, class or function in isolation
- Acceptance Tests: testing the application as a whole
- Integration Tests: testing the correct interaction with other applications and services

In addition, automated testing is also well-suited for testing several non-functional requirements, for example performance and security. Automating testing in most cases does however not mean that manual testing is completely removed from the testing process, but

rather that it plays a smaller role in the overall testing process. Exploratory Testing and User Acceptance Testing often remain manual. For more details on a balanced test approach and division between automated and manual tests we refer the reader to the 'Practical Test Pyramid' article by Martin Fowler [11] and also figure 5 which is further build upon the 'Practical Test Pyramid'.

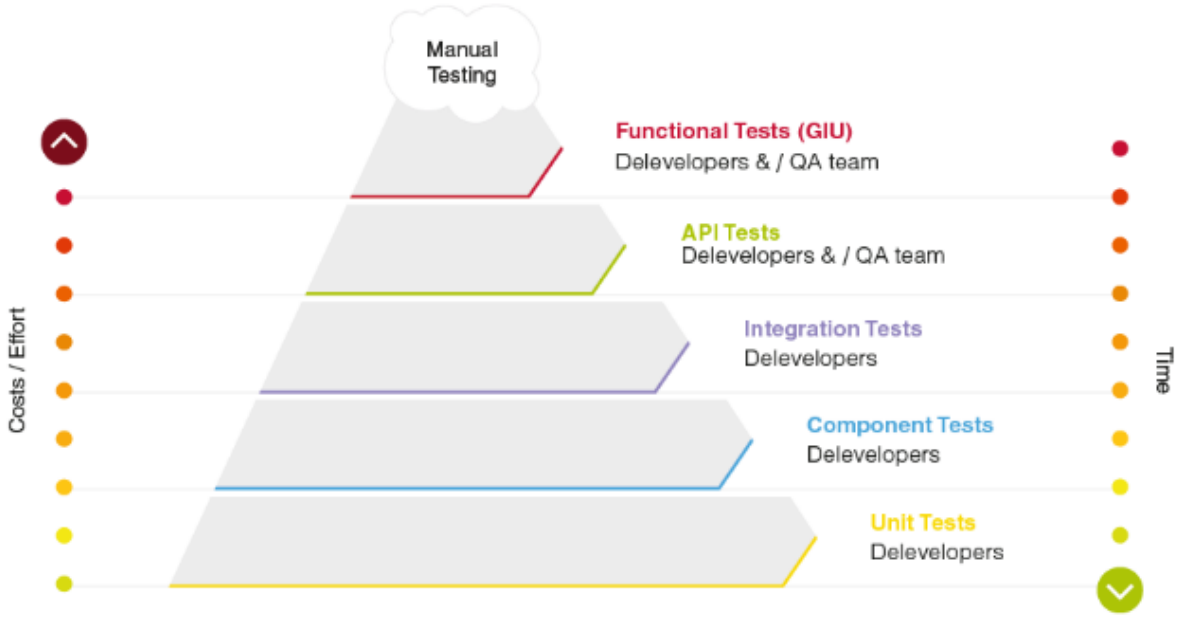

Figure 5 The ideal test pyramid [43]

From DevOps to DevSecOps: a natural evolution

An addition to our initial study report is this chapter on DevSecOps. This is not without reason, because Cyber Security is high on the global agenda. A report from World Economic Forum (WEF) and S&P Global reveals that cybersecurity-related risk consistently ranks among the top global concerns. For example, the WEF's 'Global Risks Report 2024' ranks 'cyber insecurity' as the fourth most severe risk in the short term. S&P Global states in their 'Top Geopolitical Risks of 2025' that "cyber-attacks are a growing geopolitical risk, becoming larger, more intricate, and more relentless." International organizations such as the United Nations (UN) also emphasize the growing importance of cybersecurity in their thematic reports and global policy recommendations.

Given the growing importance of cybersecurity, it's only logical that DevSecOps has emerged as the next key step in the evolution of DevOps and, by extension, in our work as IT auditors and risk professionals. Security has always been part of DevOps; the difference now is that it's taking on a far more prominent role. DevSecOps, short for development, security, and operations, represents an approach where security is not treated as a separate activity or a final step, but is instead woven into every phase of the DevOps software development lifecycle.

Before we can fully appreciate the value DevSecOps brings, it helps to take a step back and consider what DevOps has already achieved. By breaking down barriers between development and operations, DevOps transformed how software is built and delivered with a strong focus on speed, collaboration, and reliability. But despite these improvements, security sometimes remained a separate track, addressed late in the process or even after deployment. Most auditors will be able to recall at least one situation, and likely several, where security was clearly treated as an afterthought. Perhaps it was a critical (software) vulnerability discovered only after go-live, hardcoded admin credentials identified during an audit, or a secrets file that had unintentionally been committed to a public repository. These kinds of findings are all too familiar in our field. They highlight the risks of sidelining security during development and reinforce the importance of a more integrated, security-by-design approach such as DevSecOps.

DevSecOps builds upon the security foundation that should already be embedded within DevOps, reinforcing security as a shared responsibility throughout the entire development lifecycle and establishing it as a fundamental pillar. It does not replace DevOps but rather extends it. While DevOps emphasizes collaboration between development and operations and frequent delivery, DevSecOps introduces an additional dimension: security integration and secure delivery. The earlier characterization may seem overly simplistic, as security has always been part of DevOps. However, with DevSecOps, there is a much greater, more integrated, and above all shared responsibility for security, which is now recognized as not merely a technical matter but a cultural and organizational imperative.

This difference becomes particularly evident when examining how testing is approached. In a typical DevOps setup, automated testing primarily targets functionality, quality, and performance, including unit tests, integration tests, and system-level checks. These remain essential. DevSecOps, which is already integrated in some DevOps environments, introduces an additional dimension by embedding security-focused tests directly into the pipeline. Examples include security testing, vulnerability scanning, and checks for exposed secrets, hardcoded credentials, and known misconfigurations.

These practices help teams catch and address security risks early in the process, making security part of the build and delivery itself, not an afterthought or a compliance checkbox. This mindset also encourages a culture where security is everyone's responsibility, embedded in the way DevSecOps teams work together. As described in chapter 2.2, Agile development is not just about tools and processes, it's about adopting a mindset of continuous improvement and collaboration. DevSecOps builds on that foundation by making security a shared responsibility throughout the entire development lifecycle.

For us as IT auditors and IT risk professionals, the evolution from DevOps to DevSecOps is, at its core, not a major technical transformation. Rather, it represents a shift in focus. Security is, and always has been, part of the (DevOps) software development process. What changes now is that it becomes more deeply integrated across all stages, proactive, and a shared responsibility.

To summarize:

1. **Security is integrated, not isolated:** DevSecOps embeds automated security checks directly into CI/CD pipelines.
2. **Security becomes proactive:** Vulnerabilities are identified and addressed early, reducing both risk and remediation costs.
3. **It drives cultural change:** Teams adopt a mindset where developers, testers, and operations all take ownership of security, not just the security team.

DevSecOps is therefore not just a technical enhancement. It represents a natural evolution in how security is approached, implemented, and how it can be audited.

To make this rather extensive introduction to DevSecOps more concrete within the context of NOREA's DevOps Control Framework, when reviewing the controls in the DevOps Control Framework, security appears in several testing-related controls. However, it is most explicitly and concretely addressed in Control 11, which focuses specifically on security testing. This includes key components such as threat modeling, vulnerability scanning, static code analysis, dependency checks, and penetration testing, each an essential part of the "Sec" in DevSecOps.

It's important to note that not all teams are at the same stage in their security journey. Some may not yet need to be, depending on their context. The framework's maturity levels reflect this variation well. Level 1 begins with basic security testing and follow-up, while Level 5 involves continuous learning, evaluation, and iterative improvement of security testing practices.

3. DevOps in control

Based on our research and as introduced in the preceding paragraphs we advise a 3-step approach for auditing DevOps environments:

1. Determining the software development methodology or principles being used
2. Cultural maturity assessment
3. Control assessment

There are many software development methodologies, best practices and approaches. For most, if not all, traditional software development methodologies there are several control or compliance frameworks available. IT auditors are mostly familiar with the Waterfall software development methodology and therefore, most of the control frameworks used by IT auditors are Waterfall based. However, IT auditors currently face a misalignment between their control frameworks and the development practices used by the organizations. An increasing amount of organizations are using modern approaches such as Agile or DevOps or a mix between Waterfall and (parts of) Agile. We present in this chapter a combined audit approach for both Agile and DevOps.

3.1 Determining the methodology being used

There is a lot of confusion about the application of Agile and DevOps, because they are often claimed to be applied when adherence to their principles is only partly fulfilled. This is often the case when an organization is using a phased approach in the shift towards an Agile and DevOps way of working. When performing an audit under these circumstances, it is crucial to apply an appropriate control framework. We therefore advise the IT auditor to first determine which is currently the dominant approach being used and then apply the proper controls based on that methodology or practice.

One of the core distinctions between the different practices is the delivery frequency (the speed in which changes are deployed in production). Because this metric provides the most accurate indication of the dominant software development approach, we apply this metric to determine the software development method being used. The table below provides an indication of the delivery frequencies that are typically associated with each approach.

Delivery frequency	Methodology/practice	Description
Quarterly or less	Waterfall	The software development is done in phased steps leading to large planned software releases.
Monthly	Agile (principles and procedures)	The software development process follows an Agile approach, but deployments are still performed manually.
(bi-)weekly	Agile+	A CI/CD pipeline is implemented and used to deploy software to the production environment, but manual steps are still required.
Daily or more	DevOps / Continuous Deployment	Every change that is accepted is automatically build, tested and delivered by the automated delivery pipeline and possibly also deployed to the production environment.

Table 1: Guidance to determine software development method

For more details on distinctions or best practices associated with the different approaches, see the State of DevOps report 2018 [3] and DevOps Topologies [24].

3.2 Culture maturity assessment

COBIT cautions that "Culture, ethics and behavior of individuals and of the enterprise are often underestimated as factors in the success of governance and management activities". In the COBIT 2019 model we can see that culture, ethics and behavior is also one of seven components required for an effective governance system. Currently, COBIT is still the most used framework for IT auditors who use the complete, or tailored components of the model to assess the (IT) governance system of organizations. COBIT already confirms that to gain a complete insight in the working of the governance model at an organization, the IT auditor should include the assessment of the organizational culture into audit approach.

The delivery frequency is the key indicator to determine the dominant methodology being used, because it is simply not possible to achieve the higher delivery frequencies without having implemented most of the DevOps technical principles. However, from our definition of DevOps in paragraph 2.3, it can be concluded that cultural principles play a key role in maintaining a sustainable DevOps team next to the technical principles [34].

It appears that there is no common understanding about what culture is. In this guide, we choose the definition by Westrum. Westrum defines culture as that set of processes that shapes organizational response to the challenges and opportunities that organizations face [34]. Westrum explains that with 'response' he refers to the coherent patterns along which individuals and the team respond and these patterns refers not only to the action but also to the thoughts and emotions of the individuals [34]. Based on this definition, the culture of an organization can be seen as analogous to the personality of an individual.

What makes a good (DevOps) culture?

Google started project Aristotle with the goal to identify the aspects that make a team effective at Google. The project identified five factors that really mattered. In order of importance these are [35]:

1. Psychological safety
2. Dependability
3. Structure & Clarity
4. Meaning
5. Impact

The results of the project including guidance for improving each of above factors are published by Google [35].

Another model on culture can be derived from research performed by Westrum [34], which is also the model used in the State of DevOps studies. Westrum' s model consists not on a set of factors or capabilities but contains a list of 6 questions. These questions may be slightly altered to fit a particular organizational context, but only minor changes should be applied [36]. Copyright restrictions prevent us from listing these questions in this study report; we refer the reader to Westrum' s paper [36].

A third example is derived from ISACA who has also published a list of the most important factors to make DevOps teams successful [37]. These are:

- Trust
- Transparency
- Accountability
- Communication
- Mutual recognition
- Ability to learn from peers
- Ability to teach team members
- Cultural awareness

How to perform a culture assessment

Both Google's project Aristotle and Westrum' s research conclude that an organizational culture is a perceptual measure, which is hard to describe and therefore best captured using survey methods. Example of survey statements of Google based on the five factors are:

- Psychological safety - "If I make a mistake on our team, it is not held against me."
- Dependability - "When my teammates say they'll do something, they follow through with it."

- Structure and Clarity - “Our team has an effective decision-making process.”
- Meaning - “The work I do for our team is meaningful to me.”
- Impact - “I understand how our team’s work contributes to the organization's goals.”

The development of a culture assessment model is not part of this study report, but we refer the reader to appropriate assessment tools that are already available for DevOps. Examples of tools available:

- The DORA assessment tool [12], specifically the Capabilities part that focuses on the cultural readiness. This model is partly based on the Westrum model.
- The Microsoft DevOps assessment tool [13], specifically the Culture section.

It is recommended assessing the cultural maturity of the team (as required for a successful implementation of DevOps) during the audit, in order to be able to formulate, together with the assessment of the technical controls, a more justifiable audit conclusion.

3.3 Control assessment

The heart of DevOps is the Delivery pipeline that integrates the Build, Test and Delivery phases of the software development process. It consists of a dedicated implementation stream per application, based on tooling for version control, build automation, provisioning, configuration management and deployment. In figure 5 an example of the Delivery pipeline of a DevOps team is visualized [44]. The numbered controls of the control framework presented in paragraph 3.4 have been inserted in this figure to provide better insight on the ‘location’ of these controls within the Delivery pipeline.

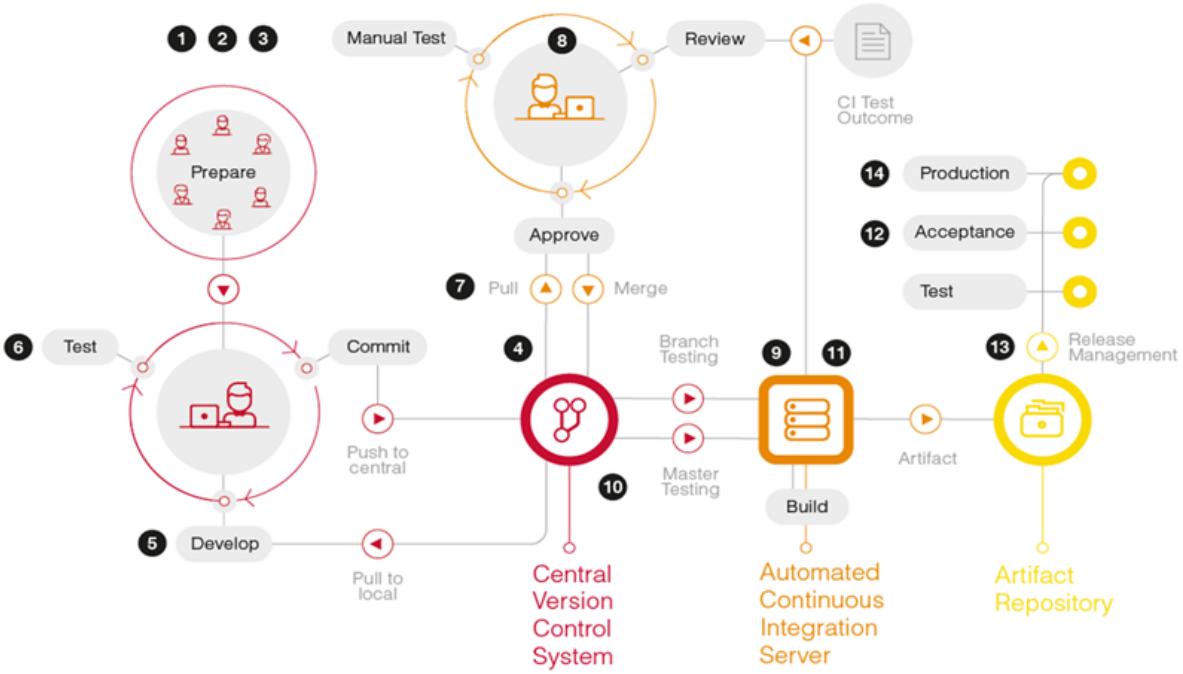

Figure 6: Example of a Delivery pipeline at Schuberg Philis [44]

IT General Controls (ITGC)

There are five domains of ITGC controls identified as the essential areas of the IT "space" that should be examined, even if only briefly, by the auditor as areas of IT that potentially introduce risk to the financial statements i.e., the risk of material misstatement (RMM) [22]. There is no common ITGC framework available, however auditing literature such as the Statements on Auditing Standards (SAS) No. 104-111 has summarized the need for these five domains to be considered [29]. ISACA publishes leading best practices and frameworks for information services and has developed a guideline which includes the minimum five ITGC domains and controls [22]. This overview of ISACA is presented in table 2, which includes their mapping to the COBIT 4.1 framework. In the table a new column has been added to include the references to newest COBIT release (COBIT 2019).

For applications within the scope of the IT audit that are managed by DevOps teams, the controls primarily affected are those within the Change Management domain, although other domains (e.g., Information Security) may also be impacted. While the control objectives themselves remain unchanged, the implementation of controls—and consequently the testing approach—differs in a DevOps context. This has been confirmed by ISACA in the COBIT 2019 Framework (see paragraph 1.2).

This study report presents a control framework for auditing DevOps environments – specific Change Management - in paragraph 3.4. It is recommended that auditors assess the controls outlined in this framework instead of the traditional controls within the Change Management domain. The remaining controls presented in Table 2 remain applicable and should be assessed in accordance with standard audit procedures.

In paragraph 2.3 we introduced the Shared Services teams which are often found in more mature DevOps organizations [3]. When assessing an IT environment in which the use of Shared Services teams is made, it is expected that the ITGC controls will need to be assessed for these Shared Services teams as well, because the management of these tools has a direct impact on (the integrity of) the application environment maintained by the Agile and DevOps teams. Furthermore, if the Shared Services teams also operate based on Agile and DevOps principles, we also suggest the auditor to assess the Change management domain based on the Agile and DevOps controls presented in in paragraph 3.4.

Domain	Controls	COBIT 4.1 reference	COBIT 2019 reference
IT entity-level controls	IT governance IT operations management	PO domain (PO01-10) DS1 Define and manage service levels DS3 Manage performance and capacity DS6 Identify and allocate costs DS7 Educate and train users DS8 Manage service desk and incidents	APO domain (APO01-APO014) DSS01 Managed operations DSS02 Managed service requests and incidents DSS03 Managed problems MEA domain (MEA01-MEA04)

Domain	Controls	COBIT 4.1 reference	COBIT 2019 reference
		DS9 Manage the configuration DS10 Manage problems DS11 Manage data DS12 Manage the physical environment DS13 Manage operations ME domain (ME1-4)	
Change management	Changes to software/programs Changes to infrastructure	AI domain (AI1-7)	BAI domain (BAI01-BAI11)
Information security	Physical and environmental controls Logical access controls	DS5 Ensure systems security	DSS05 Managed security services
Backup and recovery	Backup of data Business continuity planning (BCP) Disaster recovery planning (DRP)	DS4 Ensure continuous service	DSS Managed continuity
Third-party IT providers	Outsourced IT Vendor management ISAE 3402 audits	DS2 Manage third-party services	APO10 Managed vendors

Table 2: ITGC areas according to ISACA [22]

The concept of Shared Services teams in relation to Agile and DevOps teams is referred to by Gartner as shifting 'Up the stack' [30] since the Agile and DevOps teams managing the business applications are more and more only operating on the upper half domain of the IT stack. The lower half is 'outsourced' to Shared Services teams [3] [30]. As stated in the State of DevOps report, this results in normalization and standardization of the stack and use of best practices, which is imperative for the further maturation of DevOps within the organization [3].

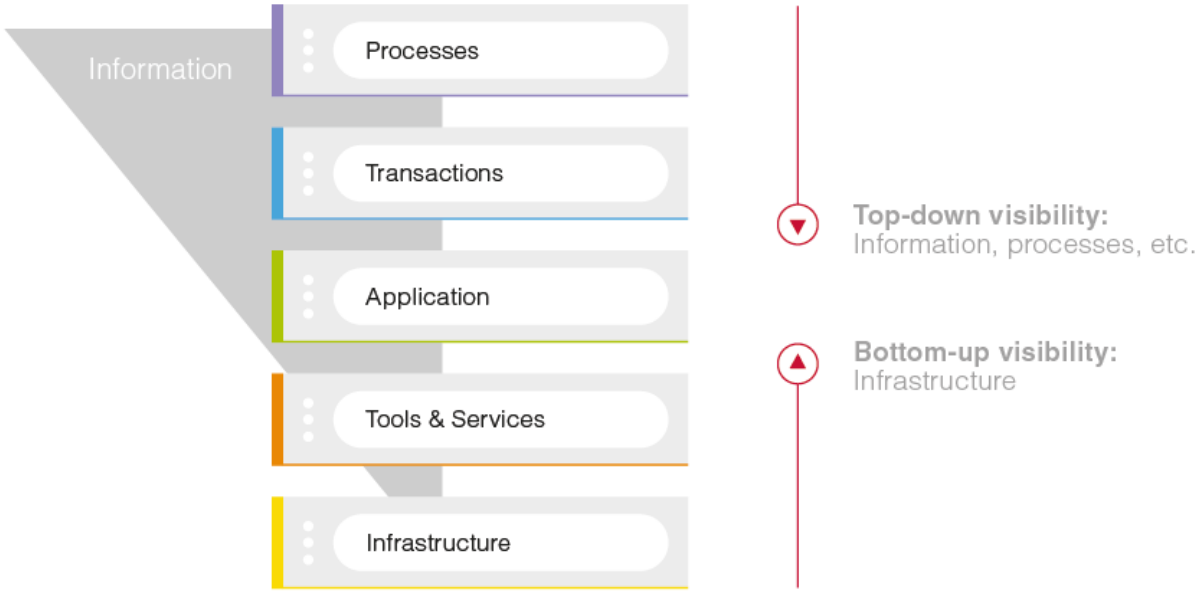

Figure 7: IT stack divided between DevOps and Shared Services teams [30]

Testing approach (system-driven versus sample-based)

From audit guidelines such as those from the American Institute of Public Accountants (AICPA), we can determine that there are five types of test procedures that can be applied during the IT audit. These test procedures need to be applied to be able to form an opinion on the suitability of the design and the operating effectiveness of controls during the period under review. The AICPA guidelines also state that the controls need to be tested by applying a variety of testing procedures. These five test procedures are (in order of complexity from lowest to highest):

1. Inquiry: based on interviews with appropriate management and staff about the controls.

2. Observation: observation of the presence of the control (e.g. a physical control such as a security camera).

3. Inspection of evidence: collection and review of documentation based on a sample size. If, during testing, the auditor encounters an error the sample, they can expand the sample size and conduct further testing or perform additional tests.

4. Reperformance: the auditor manually reperforms/executes the control to validate the output of e.g. an automated system generated calculation or in case of an automated control the inspection of just one event would be completed.

5. Computer Assisted Audit Technique (CAAT): method to analyze large volumes of data or all transactions or events executed instead of a sample size often performed with the use of software.

The use of Inquiry should be combined with other test procedures, specifically Observation, Inspection or Reperformance. The ideal test approach is testing the controls as automated controls based on the Reperformance test procedure, because then it is sufficient to test one event only instead of a sample consisting of multiple events. This approach is more effective

because it provides assurance that all events are properly executed according to the control objective and also more efficient because it requires less effort to test.

In the preceding paragraphs several concepts have been introduced. The most important concepts from an audit point of view are:

- Application of a VCS (Version Control System)
- Application of IaC (Infrastructure as Code)
- Application of CI (Continuous Integration) principles
- Application of CD (Continuous Delivery) principles
- Application of Continuous Deployment principles
- Management of (Delivery pipeline) tools based on standardized best practices by Shared Services teams (shifting 'Up the stack')

Depending on team maturity, a subset of these concepts or all of them may be applied by Agile and DevOps teams. The more concepts are applied, the higher level of automation will be achieved by the team. The implementation of these concepts is not possible without appropriate tooling (e.g., Continuous Delivery requires a CD tool) [23]. High level of automation in the Delivery pipeline and automation of the controls makes it possible for the auditor to justify the application of a system-driven audit approach, which is the most efficient and effective approach because the testing of one event only is required (Reperformance). Some examples of controls where testing one event (Reperformance) can be performed are:

- reviewing the peer-review process (pull and merge request performed by two different team members) system parameters of the tested system by tracing through one transaction.
- reviewing the query or code of the underlying peer-review check.

There are several pre-requisites that must be fulfilled for applying a system-driven audit approach. These are:

- Effectiveness of the ITGC controls of the environment in which the automated controls run (e.g., access and change management controls).
- Completeness and accuracy of all changes that directly or indirectly impact the configuration settings of the automated controls and their proper assessment and approval.

Challenges with the application of a system-driven test approach in Agile and DevOps environments

In paragraph 3.2, it has already been stated that Agile and DevOps are not fixed methodologies, but a way of working based on a set of common principles aimed at continuously improving the value, quality and speed of the delivery pipeline. Also, one of the fundamental principles is the ambition to keep improving (principles 9 and 12). This implies that a team will start as it sees fit (and considers achievable) and over time will gradually progress and increase the level of automation/use of practices. However, in order to apply a system-driven audit approach the level of automation should be relatively consistent

throughout the year and should include all the key controls. Unfortunately, this maturity stage has not yet been achieved by most Agile and DevOps teams. The State of DevOps survey results show that only 11% of respondents report a highly mature DevOps practice [3].

Another challenge to be addressed is that in most Agile and DevOps teams, team members have access to the configuration settings of the Delivery pipeline tools being used. Assessment of these access rights is part of the ITGC control assessment performed by the auditor, because establishing appropriate access rights is a pre-requisite for reliance on automated controls. If team members can change Delivery pipeline tool configuration settings at will, it is most likely that ITGC controls regarding access rights and proper segregation of duties will not be complied with, and that the prerequisites for reliance on a system-driven audit approach will not be met. Establishing Shared Services teams will help in achieving this prerequisite, because the management of configuration settings is then shifted from development team members towards the Shared Services teams.

It is also observed that, possibly due to the principles for constant improvement (principles 9 and 12), team members prefer access to the configuration settings of the Delivery pipeline tools, in order to experiment with different settings to find the most effective and efficient ones. Another important reason for team members to have access to certain features is that in emergency change procedures it is technically required to be able to change settings in the pipeline or override version control enforced protection. In the State of DevOps survey results it is confirmed that teams often start/stop several practices along the way [3] and that a stable practice is only observed in the most mature teams [3].

Introducing the FEAT-approach for control assessment

The control framework developed in this study report is applicable for Agile, DevOps and various hybrid combinations that are in common use. However, it is imperative that the IT auditor determines whether a specific control qualifies as an automated control or (partly) depends on manual activities (see prerequisites in preceding paragraph). By knowing the stage of application of the key principles and tools within the development team, the auditor can tailor the control framework towards his use. In the last column of the control framework we have indicated which controls have the potential to be tested based on a system-driven test approach (qualified automated controls). As can be seen, most of the controls have that potential but as explained in the paragraph above, it is unlikely for most teams to achieve a sufficient level of maturity to make this possible.

An alternative to sample-based testing is the Full population & Exception Analysis Testing (FEAT) method [46]. As the name indicates, this method is based on full population testing instead of sample-based testing. This method is suggested for use until the prerequisites for automated controls to enable a system-driven audit approach are met.

The FEAT method consists of the following steps:

1. Define a risk-based overview of the key controls present within the delivery pipeline.

2. Create reliable population overviews of all events related to the key controls based on complete and accurate population lists.

3. Define success/fail control logic for the control (e.g. merge requests are performed by a different team member).

4. Automate testing of the control logic and execute the test on the full population.

5. If exceptions are reported, perform an analysis of all exceptions.

6. Provide an overall conclusion based on performed analysis (e.g. effective/ineffective).

This approach provides a high level of transparency on the key controls within the Delivery pipeline. Creating transparency is seen as the main driver to achieve trust which is often still perceived as low regarding Agile and DevOps. The importance of trust is also confirmed and further elaborated upon in the 'Building digital trust' report by PwC [32]. Within teams, trust has been associated with improvements in communication, teamwork and superior levels of team performance [33]. Also, Patrick Lencioni states that one of the core contributors to a team's inability to achieve goals is due to lack of trust. He introduced the *5 Dysfunctions of a Team model* of which the most important dysfunction is called 'Absence of Trust' [31].

The application of the FEAT model can significantly help in reducing the lack of trust by:

- Providing 100% insight in the effectiveness of key controls (through full population testing)
- Providing insight in the remaining risks (through the exception analysis)

If control failures are identified, it is expected that teams will improve their control performance levels (in line with the continuous improvement Agile principle 9 and 12) and by doing so, increase the level of confidence put in their software delivery processes. The maturity levels provided can guide the organization in defining the improvements needed.

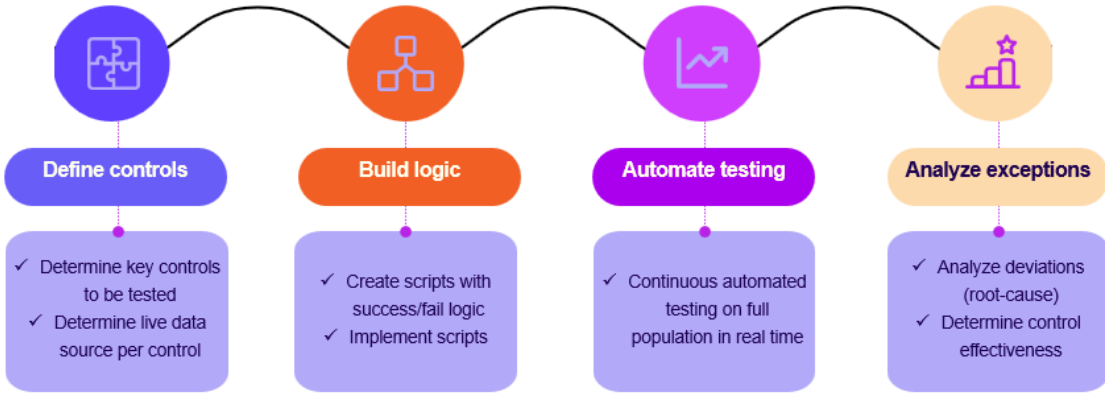

Figure 8: Summary of the FEAT method

3.4 The DevOps control framework

Version control

Version	Changes	Date
V1.0	Initial version	1-9-2019
V2.0	Improved Control description (column C) Control Assessment (column D) for all controls Merging related controls Addition of maturity levels for all controls	1-1-2026

Feedback & questions
Feedback and questions can be sent to:

- Sandeep Gangaram Panday (chair) - sandeep@brightlyn.nl
- Edwin Galama - E.Galama@cjib.nl
- Pieter Jolen - p.jolen@vanlanschotkempen.com
- Zubair Yaseen - Zubair.Yaseen@abnamroclearing.com

- Jean-Jacques Bistervels - jean-jacques.bistervels@bovemij.nl
- Than Son Nguijen - ttsonnguijen@gmail.com
- Paul van Kemenade - Paul.van.Kemenade@rabobank.nl
- Boris Cuijpers - Boris.Cuijpers@rabobank.nl

Link to Excel version
https://www.norea.nl/organisatie/kennis-en-werkgroepen/kennisgroep-software-development

How to use the framework
In line with best practices for internal control, it is recommended to conduct a risk analysis prior to selecting controls. This analysis clarifies which risks are relevant within the context of Agile/DevOps and forms the basis for defining concrete control objectives. Based on these objectives, appropriate control measures and test criteria from our framework can then be selected.

In addition, it is important to explicitly address organizational preconditions and policy aspects. Examples include:

- Ensuring well-functioning Scrum teams by focusing on desired behavior, knowledge, and competencies.
- Establishing policies and descriptions regarding access and authorization for the tooling used.
- Documenting working methods for Scrum/DevOps teams, including rules and agreements about the CD pipeline.
- Defining procedures and controls (segregation of duties) for OTAP environments within the CD pipeline.
- Including the CD pipeline and tooling in the CMDB for version, license, and configuration management.
- Creating test policies with quality requirements for different test types (integration, user acceptance, etc.), including logging, review, and approval.
- Specifying requirements, tooling, and test environments for integration and user acceptance tests, and recording results and approvals in VCS or alternatives.
- Making arrangements for vulnerability scans, security scans, and penetration tests on tooling, including prioritization, logging, and follow-up.
- Performing monitoring scans (e.g., via SIEM) on logging from the CD pipeline, focused on specific risks and compliance requirements.

These organizational measures are essential for effective control and serve as preconditions for the successful application of the control framework.

Maturity model

The NOREA DevOps framework introduces maturity levels for each control to help organizations assess their current position. Each level provides criteria and practical examples to determine where an organization stands and the actions required to advance. These maturity levels are based on the DNB Maturity Model as described in the DNB Good Practices for Information Security[6].

[6] https://www.dnb.nl/media/vskni24i/good-practice-ib-2023.pdf

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
1	Prepare	The team has selected and formally documented an Agile methodology (e.g., Scrum, Extreme Programming, SAFe) to guide its planning, requirements analysis, and delivery processes including the relevant guardrails for compliance, quality and security. Roles and responsibilities are defined and assigned in accordance with the chosen methodology, and backlog items (covering both functional and non-functional requirements) are continuously refined, prioritized, and managed with input from all relevant stakeholders.	1. The team has selected a suitable Agile methodology aligned with organizational requirements and this methodology, including any required guardrails (e.g., for compliance, quality, or security), is formally documented and accessible to all team members. 2. Based on the selected Agile methodology roles and responsibilities such as Product Owner, Scrum Master, team members, and relevant operational or security roles are assigned and documented. Verify that, based on described roles and responsibilities "Definition of Ready" and "Definition of Done" are clearly established, documented and applied for all backlog items, reflecting the needs and input of all relevant parties. 3. Ensure that processes exist and are followed for capturing, refining, prioritizing, and managing both functional and non-functional (e.g., security, compliance, operational) requirements in the backlog, with input from all relevant stakeholders including operations and security. 4. Confirm that there is ongoing involvement of business, development, operations, and security stakeholders in backlog grooming and sprint/release planning, and that feedback from past iterations and production operations is used to continuously improve planning and delivery processes.	BAI02.01 Define and maintain business functional and technical requirements BAI03.09 Manage (changes to) requirements BAI03.12 Design solutions based on the defined development methodology	1. Informal (use) of Agile-development methodology. The organization is familiar with the Agile approach. 2. Product owners have informal role, in most case regular management function fulfils this role. Documentation as DoR / DoD used ad hoc, in informal way. 3. Stakeholders requirements are partly considered in a informal way 4. Backlog management, user stories, DoD and continuous prioritization are not used yet, but could be implemented. Only a list of work to do exists	1. An informal Agile Methodology is available. 2. Informal role of Product owner and team roles exist. Dod is partly used. 3. Stakeholders requirements are considered in a informal way 4. Backlog, user stories, DoD and continuous prioritization by de Product owner is informally used	1. DevOps teams have selected and implemented a suitable Agile methodology which is properly approved and documented. 2. Product owners and teams are assigned and responsible for selected (business) services and/or products. The Product Owner is responsible for risk management tasks. Requests for functional or non-functional requirements are initiated by business stakeholders (customers) are reviewed and documented. The Definition of Ready is defined. 3. Stakeholder requirements and acceptance criteria, are considered, captured, prioritized and recorded before implementation. 4. Use of backlog system/tool. Backlog items are categorized, prioritized and reviewed by Product Owner. Definition of Done is determined.	1. the use of the selected Agile-Methodology is measured 2. The Product owner and team functioning is measured and reported. 3. Stakeholders requirements are evaluated in a dedicated session and systematic way (including client/stakeholder). 4. Backlog, user stories, DoD and continuous prioritization by de Product owner conform de Agile Methodology is applicable and evaluated in the retrospective	1. The usage of the Agile Methodology is systematically evaluated and improvements are documented and monitored closely. 2. The product owner and team functioning is improved based on the periodical evaluation (by HR and management). 3. Improvements stemming from the stakeholder evaluation session are systematically followed-up, prioritized and included on the backlog. 4. Quality of the backlog, user stories, DoD and prioritization by the Product owner is evaluated continuously; improvements identified are followed up accordingly.
2	Prepare	Develop and document designs (incl. architectural diagrams) for the	1.Based on the service portfolio and the activities	BAI03.01 Design high-level solutions	1. Designs are not available.	1. Designs are incidentally available, but	1. Based on the service portfolio and the activities on	1. Designs are formalized and available and	1. Designs are formalized and available and evaluated in the

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
		solutions in terms of technology, business processes and workflows. Ensure alignment with the IT strategy and enterprise architecture. Reassess and update the designs when significant changes occur during detailed design or building phases, or as the solutions evolve. Stakeholders actively participate in the designs and version updates are agreed/approved based on chosen agile methodology.	on which team members are working identify whether for the solutions in scope designs are available (including architectural diagrams). The level of detail maintained should be in line with the development method selected and appropriate for the solution. 2. Validate if all relevant roles are providing input on the designs while ensuring proper stakeholders are involved. 3. Ensure that the supporting (IT) systems for the solution development are properly documented (including the respective flow/interaction between the systems) throughout or after completion of the solution. Changes to e.g. the build street need to be documented including keeping usable logs of support systems/methods that are not used anymore.		2. Unsure whether all relevant roles are providing input 3. Supporting IT systems (e.g. CD-pipeline/Tools etc.) is not documented or partly concept-documented.	not structurally and as part of a formal approach. 2. All relevant roles are informally requested to provide input for designs. 3. Supporting IT systems (e.g. CD-pipeline/Tools etc.) are informally used to support design activities.	which team members are working identify whether for the solutions in scope designs are available. The level of detail maintained should be in line with the development method selected and appropriate for the solution. 2. All relevant roles are identified and provide input for the designs while ensuring proper stakeholders are involved for appraisal and approval. 3. Supporting (IT) systems for the solution development are properly documented (including the respective flow/interaction between the systems) throughout or after completion of the solution. Changes to e.g. the build street are documented, including keeping usable logs of support systems/methods that are not used anymore.	compliance is measured in the retrospectives 2. Input from involved roles and stakeholders is tracked and measured. Approval and decline frequencies by stakeholders are recorded. 3. The overview of supporting IT systems (e.g. CD-pipeline/Tools etc.) is documented and its use measured by the owners (e.g. after each sprint).	retrospectives. Improvements are sought and implemented. 2. The list of relevant roles is evaluated periodically. Improvements are sought and implemented based on input and approval/decline figures. 3. The overview of supporting IT systems (e.g. CD-pipeline/Tools etc.) is documented and evaluated by the owners periodically (e.g. after each sprint). Improvements are sought and implemented.
3	Prepare	Procure solution components (applicable to the CI/CD pipeline) in accordance with requirements, detailed designs, architecture principles and the enterprise's overall procurement policies and procedures (considering security/privacy and compliance requirements). A periodic review of procured solutions is conducted to ensure that newly added features (such as generative AI features) still meet the enterprise's policies.	1. Identify company procurement procedures and requirements (including security, privacy and compliance) against which the candidate solutions are assessed. 2. Validate whether stakeholders are involved in the procurement process and whether the procured software complies with company requirements. 3. Validate if an overview is available of all external tools/software used (including open source) and match this with the acquisitions in an asset inventory. 4. Extend the (software) asset inventory with a SBoM (Software Bill of Materials) to ensure that per tool/application a complete list is available of the components, libraries and	BAI03.04 Procure solution components	1. Procurement procedures and requirements are not set up formally or not known 2. Stakeholders are involved on an ad hoc basis 3. An overview of external tools/software is not available	1. Procurement procedures and requirements generally known but not documented. 2. Stakeholders are identified and involved, however not documented. 3. An overview of external tools/software is available and matched with the asset inventory (e.g., CMDB) on ad hoc basis.	1. Candidate solutions are assessed against company procurement procedures and requirements (including security, privacy and compliance) 2. Stakeholders are involved in the procurement process and the procured software complies with company requirements. 3. An overview is available of all external tools/software used and matched with the acquisitions in an asset inventory. A periodic review of the tools in the asset inventory is performed and follow-up of actions is performed. 4. The asset inventory is extended with a SBoM.	1. The adequacy of the company procurement procedures and requirements (including security, privacy and compliance) and the effectiveness of the assessment is periodically measured and reviewed. 2. The list of stakeholders is verified for accuracy periodically. Whether the procured software complies with company requirements is validated periodically. 3. The coverage of external tooling and its suitability are measured. Reporting on periodic review and outstanding actions is available. 4. Periodic measurements are performed to assess completeness of the SBoM.	1. The effectiveness of the company procurement procedures and requirements (including security, privacy and compliance) and the assessment of candidate solutions are continuously evaluated and improved 2. The effectiveness of the procurement process is continuously evaluated and improved 3. Improvements for the use of external tooling are actively identified through market surveys or other means. Large releases are automatically identified and reported for timely follow-up for review against the enterprise's policies. 4. Tooling is used to maintain a complete and automated SBoM

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
			frameworks used within the tool/application. In case of acquired software this can be requested from the supplier.						
4	Develop	A central version control solution is used for all software artifacts, including application code, infrastructure code and test scripts to automate version management and ensure traceability across repositories.	1. Validate that a central Version Control System (VCS) is implemented for all relevant software artifacts, including application code, infrastructure code, and test scripts. Confirm that the VCS is properly configured to support secure, auditable version management. 2. Validate that all code and script changes are logged within the VCS, capturing at least: who made the change, what code was changed, when the change was made, reviewer comments (if applicable). Ensure that change history and log data are retained for a period defined by organizational policy and compliance requirements. 3. Validate that access rules are defined and enforced for the repositories, pipelines, and testing tools. Test that only authorized users can modify repository settings or access sensitive functionalities. 4. Validate that a clear branch policy is defined, such as requiring feature branches and enforcing peer review (the 4-eyes principle). Confirm that this policy is actively followed and enforced by the configuration of the VCS or supporting tools. 5. Ensure that exceptions to version control or access rules (if any) are documented, justified, and subject to formal review and acceptance by the relevant authority (such as the product owner).	BAI03.03 Develop solution components	1. A formal version control is not implemented 2. Code changes are not logged or logged ad hoc 3. Specific access rules are not formalized 4. The branch policy has not been defined	1. A version control system has been implemented with the proper configuration settings 2. Code changes are logged and the log data is maintained. 3. Access rules have been defined and properly implemented 4. A Branch policy has been defined and is followed	1. A version control system has been implemented with the formalized proper configuration settings 2. Code changes are logged and the log is retained in line with formal policies 3. Access rules have been defined and are properly implemented. 4. A Branch policy has been defined, is followed and verified.	The effectiveness of the measures under maturity level 2 are reviewed periodically 1. The configuration settings of the version control system are periodically reviewed. Metrics are used to measure compliance with a defined settings baseline. 2. Metrics for code changes have been set and are actively measured and reported. 3. Access rules have been defined, properly implemented and are reviewed periodically. Metrics are used to measure compliance with the policy. 4. A Branch policy has been defined, is followed and verified periodically. Metrics are used to measure compliance with the policy.	The measures under maturity level 2 are continuously evaluated and improved. 1. The configuration settings of the version control system are continuously evaluated and improved. 2. Code changes are verified and possibilities for improvement identified. 3. Access rules have been defined, properly implemented and are reviewed periodically and reviewed for improvements 4. A Branch policy has been defined, is followed and verified periodically for improvements 5. Rules for the storing and processing of sensitive information are made. A scan is performed periodically to uncover sensitive information. Possible improvements for storing sensitive information are actively identified and reported.
5	Develop	Develop solution components incrementally within isolated, secure	1. Verify that a documented coding standard addressing	BAI03.03 Develop solution components	1. At best, an informal coding standard with	1. Basic coding principles with clear guidelines are	1. A coding standard is available and implemented and	1. A coding standard is available and implemented.	1. A coding standard is available and implemented.

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
		development environments that comply with company standards and security policies. Develop solution components progressively in a separate, secure environment, in accordance with company standards.	language, style, and secure development practices exists and is accessible to the team. Confirm that code is automatically checked for adherence using tools such as linters or static code analyzers, and that identified violations are remediated. 2. Ensure that secure coding policies and guidelines are implemented and actively enforced via integrated development tools (e.g., linters, static analysis tools). Confirm that any policy violations are tracked and resolved before code advancement. 3. Validate that the selection and use of external software components, including libraries and frameworks, follows established guidelines covering security, support, and licensing. Check that decisions regarding component selection are documented for traceability. 4. Ensure that processes are in place to prevent the inclusion of unencrypted sensitive information (such as passwords, access keys, and secrets) in source code repositories. Where possible, automated scans are performed to detect any secrets or sensitive data in the codebase and such findings are promptly remediated.		guidelines is in place but ad-hoc used. Coding is done according to insights and experience of development staff. 2. Limited use of (secure) coding policies and coding tools/framework. 3. Limited guidelines in place for selection of external software components and this is ad hoc and driven by individual opinion and/or need. 4. No formal agreements for the storing and processing of sensitive information have been made	partly implemented based on a coding standard. Guidelines are not included in a policy document. 2. Use of (a selected number of) secure coding policies, but partly implemented in tools/frameworks dependent on the maturity of the development team. 3. Selection of external software components (including software libraries) is performed based on partly implemented policies and based on a per case evaluation. 4. Rules for the storing and processing of sensitive information are made.	included in a policy document that is communicated through the relevant parts in the organisation. The standard is reviewed periodically. 2. Implementation of a selected number of reliable and secure coding policies and enforcement of these policies in the pipeline by use of coding tools/frameworks integrated in the programming editor. 3. External software components (including software libraries) are selected based on agreed upon guidelines that have been implemented throughout the organisation. These guidelines are aimed at compliance on security / maturity requirements. 4. Rule for the storing and processing of sensitive information are made. A scan is performed to uncover sensitive information	The coding standard contains guidelines for the use of (a) programming language(s), programming style, practices and methods. The standard is known by the relevant parts in the organisation and its application is measured against, for example, industry standards. 2.Selected (secure) coding policies are implemented and enforcement of these policies by use of coding tools/frameworks is integrated in the programming editor (e.g. eslint, pylint, pep8, etc.). Metrics for coding quality have been defined. 3. External software components (including software libraries) are selected based on agreed upon guidelines throughout the organisation. These guidelines are aimed to ensure compliance with security / maturity requirements. The application guidelines and the selection of external tools is measured and minimum metrics defined. 4. Rules for the storing and processing of sensitive information are made. A scan is performed periodically to uncover sensitive information. The adherence to the agreement is measured.	The coding standard contains guidelines for the use of (a) programming language(s), programming style, practices and methods. The standard is known by the relevant parts in the organisation. The application is continuously evaluated and tested as part of good risk management practice. 2. Ensure the use of selected (secure) coding policies and enforcement of these policies by use of coding tools/frameworks integrated in the programming editor (e.g. eslint, pylint, pep8, etc.). The implemented coding policies are continuously reviewed and improved upon when possible. 3. External software components (including software libraries) are selected based on agreed upon guidelines that have been implemented throughout the organisation. These guidelines are aimed at compliance with security / maturity requirements. The guidelines and the selection of external tools are continuously reviewed and updated.
6	Develop (test)	Incorporate risk-based automated testing as an integral part of the development workflow. Developers create and maintain automated test cases (unit and/or component tests) that demonstrate the code functions as intended and adheres to approved design and coding standards. All automated tests are stored in version control alongside the codebase to ensure continuous	1. Verify that a risk-based testing approach is defined and applied, requiring at minimum automated unit or component testing of all code changes. 2. Confirm that the team has established, and documents, explicit requirements for test coverage (e.g., percentage per module or feature),	BAI03.07 Prepare for solution testing BAI07.04 Establish a test environment BAI03.08 Execute solution testing	1. Testing is an ad hoc approach and minimum requirements have informally been set. 2. Ad-hoc measures or requirements for test coverage are set.	1. A (risk based) test approach is partially implemented with a minimum level of tests. 2. Requirements on risk allowance and test coverage, or equivalent industrial standard, are set but partially implemented and its application is not	1. A risk based test approach is documented and implemented and requires the execution of at least Unit Testing or Component Testing of the code changes made by the developer. 2. Requirements on risk allowance and test coverage, equivalent industrial standard, are set and implemented and are measured and/or reported.	1. A risk based test approach is documented and implemented and requires the execution of at least Unit Testing or Component Testing of the code changes made by the developer. The execution of the approach is measured and periodically verified. 2. A test approach is implemented and includes	1. A risk based test approach is documented and implemented and requires the execution of at least Unit Testing and Component Testing of the code changes made by the developer. Identification of possibilities for structural improvement is integral to the test approach. 2. A test approach is implemented and includes

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
		validation and enable reliable testing of future changes (see control #4 as well).	aligned with industry benchmarks or internal targets. Ensure regular reviews are performed to assess test coverage evolution, adjusting thresholds or strategies based on code quality metrics, defect leakage, or evolving risk profile. Confirm the testing approach allows for variation in test depth, coverage, and review depending on the criticality of the change (e.g., standard vs. critical vs. non-critical), and that criteria for classifying change criticality are clearly documented.			documented well.		requirements on risk allowance test coverage. Risk levels and test coverage, equivalent industrial standard, is measured and meets a minimum required level of test coverage (%) per specific code module or equivalent industrial standard .	requirements on risk allowance and test coverage. Test coverage is continually evaluated for effectiveness based on a required level of minimal test coverage (%) per specific code module, equivalent industrial standard, and risk evaluation of deficiencies. Possibilities for improvement are identified and learnings from other steps must be used to improve.
7	Develop	A peer review of the code is mandatory for code changes and are executed based on the organizations code review guidelines.	1. The team has documented code review guidelines (potentially incorporating SAST requirements), based on best practices such as the Google Style Guide and, where relevant, enhanced with security checks from the OWASP Application Security Verification Standard. 2. Upon local commit, the developer pushes the code to a separate branch in the version control system (VCS). 3. A pull/merge request is created and both peer review by another developer and automated quality checks (e.g., build status, static analysis, unit tests, SonarQube gates) are triggered and enforced by the VCS or pipeline. See also control #8, 9, 10, 11 and 12 on testing. 4. The merge into the main branch is only permitted after all peer review comments are resolved/approved and automated quality checks pass. Failed checks block the merge; the developer is responsible for addressing	BAI03.08 Execute solution testing BAI03.03 Develop solution components	1. Code review guidelines are not available 2. A branch policy has not been defined formally 3. A peer review is not performed or performed ad hoc	1. The team uses code review guidelines for their peer review, however these are informal and not completely documented yet. 2. A branch policy is defined and peer review is available in the VCS. 3. Peer reviews are performed, however not yet enforced in the VCS.	1. Formal code review guidelines are documented and available. A minimum rule set has been defined by the organisation. 2 & 3. A branch policy is defined peer reviews are configured as a mandatory activity in the VCS.	1. Based on experience from performed peer reviews the effectiveness of the code review guidelines is measured. Tooling for automated code review is used. 2 & 3. A branch policy is defined peer reviews are configured as a mandatory activity in the VCS. Metrics are defined and reported upon to identify the number of merged requests after peer review and potential exceptions.	1. Code review guidelines are continuously improved when deficiencies are identified. Tooling for automated code review is used and continuously optimised for purpose. 2 & 3. Automated alerting is in place when a merge request is performed without peer review.

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
			issues. 5. If applicable, a pair programming approach may be accepted as an alternative to formal review, provided this is documented and meets organizational policy. The process is enforced through VCS branch protection and/or pipeline settings to prevent bypassing required reviews and tests.						
8	Build	All changes to infrastructure and application code are automatically tested during the build process using integrated security tools (e.g., SAST, automated dependency scanning, OS baseline checks). Builds failing organizationally defined security thresholds are blocked from merging	1. Ensure the build process incorporates automated security scanning, including (but not limited to): - Software vulnerability scanning - Third-party (open-source) component/library scanning for known vulnerabilities and licensing/compliance issues - Code dependency scanning for weak or outdated dependencies - Operating system baseline scanning - Static code analysis (including ruleset conformance and security testing) 2. Confirm that clear, documented minimum security requirements are set by the organization, and that the build will fail automatically if these criteria are not met. 3. Verify that all findings from security scans are tracked, assigned, and remediated within timelines defined by organizational policy, with appropriate follow-up and documentation.	BAI03.08 Execute solution testing DSS05.02 Manage network and connectivity security DSS05.07 manage vulnerabilities and monitor the infrastructure for security-related events	1. The vulnerability/baseline scans and test plans during the build process regarding infrastructure/application codes, do not contain software, code, security and third party aspects. Ad hoc secure code scanning and informal use of peer reviews takes place. 2. Ad hoc rule settings on failing the build	1. The vulnerability/baseline scans and test plans during the build process regarding infrastructure/application codes, contain software, code, security and third party aspects. 2. Rules are set informally by the team on falling the build	1. During the build process, dependent on the team's risk profile, the following automated vulnerability scans are performed (not exhaustive list): o software vulnerability scanning; o third-party (open-source) component/library scanning for known vulnerabilities and licensing issues; o code dependency scanning for (weak) dependencies; o operating system baseline scanning; o static code analysis (conformance to defined rulesets and security testing). 2. A rules set has been defined by the team on failing the build (based on documented minimal requirements).	1. Execution and effectiveness of the automated vulnerability scans is measured. 2. The rules set on failing the build as set by the team are recorded and measured.	1. The automated vulnerability scans are evaluated periodically on effectiveness. Improvements are made where required. Checks are made of improving vuln. scanning tools to use in building process. 2. The rules set on failing the build as set by the team are evaluated periodically. Improvements are made where required. Root cause of software build errors lead to new or improved scanning tooling.
9	Test	After a successful build, the automated delivery process initiates integration testing of the entire code base in a production-like environment. These integration tests—whether automated or manual—validate that all components interact correctly with their dependencies and that major	1.Validate if the team has developed a documented integration test plan specifying the scope of tests, test methods, frequency, required tools, and the process for resolving test findings. Where appropriate, verify that a generic integration	BAI03.07 Prepare for solution testing BAI07.04 Establish a test environment BAI03.08 Execute solution testing	1. The integration test during the delivery process regarding changes are performed ad hoc. 2 The integration test are performed in an environment, which is not similar to the	1. The integration test during the delivery process are performed informally. 2 The integration test are performed in a test-environment for the application similar to production environment	1. The team has created an integration test plan specifying which tests, test methods, test frequency and test tools to apply for the given change, including the resolution method to apply. Where applicable a generic test plan related to the complete solution may apply instead of a test plan per	1. Performance indicators on the integration test process are recorded. 2 Compliance of the integration tests with test/QA plans is measured/recorded. 3. Test/QA plan metrics, test	1. The integration tests during the delivery process regarding changes are performed by a formalized plan. Management improves the test planning based on the performed evaluation of test results and test plans. 2 The integration test are

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
		product features function as intended.	test plan has been defined for the overall solution, or that individual test plans are used for specific changes as needed. 2. Ensure that a dedicated test environment is in place, closely mirroring the production environment in terms of configuration and integrations. To comply with privacy regulations, validate that rules and controls are established for the use of sensitive data in testing, for example by requiring anonymization or de-identification of personal data where applicable. 3. Confirm that the test plan, test environment setup, and test results are reviewed and validated with relevant business stakeholders, with the product owner acting as their proxy if required. Document stakeholder sign-off or feedback as part of the validation process. 4. Ensure that all integration test findings are logged in a centralized register or tracking system. Verify that findings are actively monitored, assigned for follow-up, and resolved in a timely manner, with clear audit trails showing status and actions taken prior to production release.		production environment. 3. Test plan, test-environment and test results are not shared/validated with stakeholders/Product owner. 4. Test findings are registered in an informal way. A structured approach, use of logging and follow up not yet in place.	3. Test plan, test-environment and test results are shared in de DevOps-team (including Product owner, e.g. not validated). 4. Test findings are registered in a database with name, date, resolve date, without tracking information.	change. 2. A test environment that is commensurate with the enterprise environment is available (i.e., production-like). However, to comply with rules established for test data that comprises sensitive data, e.g. rules that specify for which types of personal data the test data sets should be anonymized (de-identified). 3. Test plan, set-up of the test environment and the test results are validated with the business stakeholder (product owner). 4. A register or log is maintained for test findings that need to be resolved. Tracking is performed in such a way that team members can easily follow the resolution of these findings to ensure safe delivery.	environment variables and test results are recorded and measured. 4. Test findings are recorded and statistics are used.	improved based on the performed evaluations. 3. Outcomes are shared, validated and evaluated in retrospective with the stakeholders e.g. Product owner. Based on the retrospective with the stakeholders/product owners improvements are made regarding the Test plan, test-environment and test results. 4. Improvements are made for the test registration if applicable during the evaluation appears. Root causes for findings are evaluated and fixed.
10	Test	All testing scripts are developed and maintained in a version control system or versioned otherwise. This includes the test scripts for both the application code and the infrastructure.	1. Ensure tests scripts are documented to ensure all team members can follow the test progress throughout the process. 2. Where structural (testing) issues are present due to circumstances that cannot be remediated in the short term, these issues are properly documented including the cause, possible mitigation measures and suggestions for acceptance of the associated risk. These	BAI03.08 Execute solution testing	1. Verified and structured test scripts used a limited number of cases. Testing is ad-hoc and/or a structured process lacking. 2. Structural issues are followed up in an ad-hoc manner.	1. Test scripting is partially implemented and used scripts are partially stored in a managed repository. 2. Structural testing issues are addressed by ad-hoc incidents/quick-fixes and are partially documented/traced.	1. Test scripting is implemented and is part of a documented process, using a system of version control, verified & approved and also scripts are stored in a managed repository. 2. Structural testing issues are addressed in the product backlog and also centrally documented and tracked for effective and consistent follow-up.	1. Periodic monitoring and measurement of the scripting and the process is in place. 2. Metrics on Structural testing are defined and integrated in quality control. The follow up on backlog/testing issues is measured and clearly shows how delayed issues are prioritized based on time on open status in backlog/testing issues listing.	1. Test scripting is implemented and is part of a documented process, using a system of version control, verified & approved and also scripts are stored in a managed repository. Quality management continuously evaluates the scripting and the process to improve. 2. Structural testing issues are centrally documented, traced via quality control, assessed for dependencies and scheduled for later handling. The backlog/testing issues

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
			issues are proposed towards the product owner for acceptance and proper tracking and management attention.						is/are continuously monitored and clearly shows how delayed issues are prioritized based on time on open status in backlog listing. Management challenges the product owner to address unresolved issues based on risk and market priority.
11	Test	For each release identified as requiring additional security assurance—based on risk assessment, non-functional requirements, or threat modelling—targeted security tests (e.g., manual code review, vulnerability scanning, penetration testing) are performed to validate residual risks. Findings are prioritized, communicated to relevant stakeholders, and tracked to resolution.	1. Confirm that the test approach and test plan explicitly define when releases require additional security assurance, based on documented risk assessment, non-functional requirements, or threat modelling. Verify that the chosen security testing methods (e.g., manual code review, vulnerability scanning, penetration testing) are suitable for the identified risks. 2. Ensure that those performing or reviewing the security testing have relevant qualifications and experience appropriate to the risk and testing method. 3. Verify that all security findings are systematically registered, risk-prioritized, and assigned owners for follow-up. Additionally, ensure that test results and identified risks are communicated promptly and appropriately to the product owner and other affected stakeholders—including privacy, compliance, or business representatives when relevant. <u>Note:</u> Confirm that, although not mandatory, periodic validation/review of the overall security test approach is performed with experts from the central security or risk team.	BAI03.08 Execute solution testing DSS05.02 Manage network and connectivity security DSS05.07 manage vulnerabilities and monitor the infrastructure for security-related events	1. Test approaches and test plans do not or inconsistently address security aspects. 2. The qualifications of the team members performing the security testing is unknown. 3. Not all security based test findings are registered and prioritized.	1. The test approach and test plans informally / indirectly address security testing and testing frequency. 2. The team members performing the security testing have qualifications, but these are not documented. 3. Exceptions are informally prioritized and followed up, however this cannot be determined based on documentation.	1. The defined test approach and test plans contain security testing and testing frequency and is carried out according to plan. 2. Security scans are performed/reviewed by team members with the proper and documented qualifications. 3. Exceptions are well documented, prioritized and followed up.	1. The execution of security testing according to defined test approach and test plans is measured. 2. Metrics on security testing are available and reported. 3. Meaningful / relevant metrics on exceptions are available.	1. The defined test approach and test plans effectiveness is measured and evaluated periodically and improvements are identified. 2. Security testing activities are periodically evaluated and improvements identified and documented. 3. Periodically the metrics, exceptions and the process for exceptions (handling) is periodically evaluated and improved.
12	Test	User Acceptance Testing (UAT) is conducted on the finalized software build in a production-like environment, focusing on key business scenarios. Business process owners and end users actively participate in UAT to validate that	1. Ensure that UAT is performed in a production-like environment established according to organizational standards. Validate that test data used in UAT complies with	BAI07.03 Plan acceptance tests BAI07.04 Establish a test environment BAI07.05 Perform acceptance tests	1. The test environment, when available, is not fully production-like. 2. The business owner is insufficiently involved in the test process.	1. A production-like test environment has been set up. 2. The business owner is informally involved in the test process.	1. A production-like test environment has been set up and the requirements are documented. 2. The business owner is involved in the test process and	1. A production-like test environment has been set up and the requirements are documented and metrics on similarity are available. 2. The business owner is	1. A production-like test environment has been set up and the requirements are documented and periodically verified and continuously improved.

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
		the solution meets business requirements and acceptance criteria. Any exceptions or defects identified are documented, tracked, and resolved prior to go-live. (Automated) User Acceptance Testing (UAT) is performed on the created software build in a production like environment are performed, and noted exceptions are followed up. Business process owners and end users are involved in the UAT test.	privacy and data protection requirements (e.g., anonymization of any real personal data). 2. Confirm that a UAT plan is in place, defining specific business processes, key user scenarios, acceptance criteria, and roles for business process owners and end users. Verify active involvement of designated business process owners and end users in test execution. Confirm that test results and user feedback are reviewed and formally accepted by the business stakeholder(s) or product owner. 3. Ensure all UAT findings and exceptions are logged in a tracking system, prioritized, and addressed promptly. Confirm that resolution of all critical findings is completed prior to production release, and that final business acceptance is documented. Note: In a modern CI/CD approach, where the automated tests in the Unit/Component Testing (#6) and Integration Testing phases (#9) cover all business rules, UAT tests are typically only needed to cover key usage scenarios.		3. A log of test findings is not or partially available.	3. A log of test findings is available, but not always consistently updated.	the persons role is documented. 3. A log of test findings is available, complete and maintained.	involved in the test process and his role is documented. A periodical check exists to verify whether the business stakeholders executes his role. 3. A log of test findings is maintained. Metrics on log characteristics are available.	2. The business owner is involved in the test process and his role is documented. The role execution by business owners is periodically evaluated. 3. A log of test findings is maintained. The test process is evaluated periodically based on the log findings.
13	Producti on Deploy	Approved and tested deliveries are (automatically) deployed to the production environment.	1. Confirm that deployments are performed strictly according to the documented change management process, which must describe the CI/CD workflow and define change categories (e.g., standard, normal, emergency). 2. Verify that criteria for identifying and approving Standard (pre-approved/low-risk) changes are well-defined (e.g., infrastructure updates, configuration tweaks). Check if specific requirements for these	Not applicable	1. Change organization doesn't have procedure in place for their improvement and deployment process. 2. Criteria and requirements for standard changes are not (yet) defined. 3. Deployed changes are not related to the original change-request and no structural use of backlog system. 4. No fallback scenario available for deployed	1. Informal procedures for CI/CD are in place supporting structured deployment. 2. Criteria and requirements for standard changes are informally used but not always consistently applied. 3. A planning and backlog system for changes is in place but not yet used in a consistent way for managing deployment.	1. Deployment is performed based on the change management procedure describing the CI/CD process. The procedure describes the different change categories: Standard, normal and emergency changes. 2. The criteria for Standard changes and the requirements for these changes e.g. do they need to be registered in the planning tool or is tracking in the VCS sufficient, what level of automated testing needs to be performed, is peer-approval required if sufficient automated testing are available.	1. The performance of delivery process is measured on a constant base. 2. The applicable criteria and requirements regarding deployed changes are measured consistently. 3. The relation between de deployed changes and the original request is measured consistently. 4. Performance of the fall back process for failed deliveries is measured and execution of analysis is monitored.	1. Analyzing on a continuous basis of the CI/CD procedures lead to structural improvement of the change management process. 2.Criteria for standard changes are evaluated on a regular basis supporting improvement of these criteria and effective planning and backlog management. 3. All deployed changes can be linked to their respective change tickets in the backlog management system supporting continuously improvements features and

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
			changes are applicable e.g., for documenting, registering, tracking, and approving these changes. 3. Ensure all deployments are traceable to corresponding change request, incident, or feature tickets (e.g., via integration with JIRA or by referencing ticket numbers in VCS commit messages connected to deployments). Confirm linkage of deployments to user stories, defects, or incidents for improved auditability and context. 4. Check that failed deployments have a clearly defined fallback mechanism (rollback or fix forward) with pre-documented steps. If applicable, verify that failures trigger a post-mortem or retrospective review, analysing root causes and driving continuous improvement in the delivery pipeline. 5. Confirm that all code deployments are executed exclusively via automated, closed CI/CD pipelines—manual interventions or ad hoc scripts are not permitted for production deployment.		changes and failures are not analyzed. 5. Production deployments are performed manually using scripts or direct server access, without automation or traceability.	4. Fallback scenario are in most cases available for deployed changes. 5. CI/CD pipelines exist but are not consistently used; manual deployments still occur and are not formally restricted or tracked	3. The deployed changes are related to the respective change request tickets in the planning & backlog tool to allow more context for the executed changes such as linking them to feature defects, incidents or user stories. 4. Failed deliveries have clear fallback scenario (rollback / fix forward) and failures are analyzed to support optimizing the delivery pipeline. 5. All production deployments are executed via automated CI/CD pipelines, with manual interventions formally prohibited and traceable to change requests	5. Deployment automation is enforced and monitored; metrics are collected to measure compliance, and exceptions are logged and reviewed	stories. 4. All analyses of failed deliveries are evaluated on their value for improvements of fallback scenario leads and continuous improvement of the delivery pipeline leading to a decrease of failed deliveries. 5. Deployments are fully automated and continuously improved; manual access is technically blocked, and failures trigger automated rollback and root cause analysis
14	Production Deploy	Establish and maintain a monitoring approach for all business solutions and applications managed by the team, including their service delivery, to ensure alignment with, and measurable contribution to, enterprise objectives.	1. Engage relevant stakeholders (e.g., business owners, operations, IT) to define clear monitoring objectives, scope, and measurement methods for each business solution or service. 2. Verify the existence of a documented monitoring plan covering agreed KPIs/metrics, data sources, monitoring tools, roles and responsibilities, and review frequency. 3. Ensure the monitoring approach is kept up to date and that outcomes are	MEA01.01	A monitoring approach for system functioning and performance is not in place or agreed upon with stakeholders.	A monitoring approach exists and is applied consistently for some solutions and services. Procedures are generally followed, but documentation and evidence of monitoring activities or outcomes may be incomplete or informal.	A process for monitoring system functioning and performance is in place, approved and documented. Criteria for functioning and performance failure have been defined by management.	IT, development and end users record any failures and issues with (new) functionality and system performance. Satisfaction of stakeholders regarding meeting defined business and process goals is measured using at least DevOps criteria as: lead time for changes, change failure rate, deployment frequency and mean time to recovery.	Management and development teams periodically assess the outcomes of the monitoring activities and improvements are identified and defined by stakeholders.

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Maturity level 1	Maturity level 2	Maturity level 3	Maturity level 4	Maturity level 5
			periodically reviewed with key stakeholders to drive continual improvement.						

4. Conclusion

As Forrester describes it, *"Agile is the foundational pillar of the tech industry."* Backed by 95% of professionals who affirm its critical relevance to their operations, and 58% of business and technology leaders who prioritize Agile adoption, the message is unmistakable: Agile isn't just enduring, it's thriving. While its dominance remains clear, there's also a growing consensus that continuous refinement is essential to meet the evolving demands of modern business[7].

The use of these principles is also adopted within (highly) regulated environments. A specific example is the cloud.gov platform [4] of the US Federal Government, a Platform as a Service (PaaS) solution for US government agencies deployed in the AWS GovCloud region. This platform is built based on Agile and DevOps principles, while at the same time meeting the requirements of a highly regulated environment (FedRAMP and FISMA) [5].

Why, we might ask?

This is answered perfectly by Gartner. They state, because: "Every business is a digital business. Every company is a software company. The key to gaining and sustaining competitive advantage in digital business, and a role in a digital society, will be in the development and continuous improvement of new IT-enabled capabilities and services for customers" [41].

ISACA adds that DevOps is the combination of people, culture, processes, tools and methodologies that reduce risk and cost, enable technology to change at the speed of the business, and improve overall quality [37].

The application of Agile and DevOps principles and the achieved high-level of automation provides opportunities for the organization to enhance their audit approach to become more effective (higher level of assurance) and more efficient (less time). This is possible due to:

- The use of a Version Control System (VCS) and Infrastructure as Code (IaC) principles gives full insight in all changes to the application source code and infrastructure components by recording who, what and when changes occurred (paragraph 2.2).
- The application of IaC gives easier insight in the security baselines used to deploy instances. Testing of the proper application of security baselines on instances is often a very time-consuming and difficult control activity. Furthermore, the auditor can easily verify when recommended infrastructure security configuration settings have been applied whereas this is also difficult to assess when configurations are managed without application of automated configuration management [37].
- The ability to automatically execute code, vulnerability, dependency scanning tools on each change instead of the common periodic/monthly frequency and track the follow-up of relevant findings through the VCS and IaC logs instead of tickets in the service management tool (paragraph 2.3).

[7] https://www.forrester.com/blogs/amidst-the-ai-hype-agile-still-remains-relevant-in-2025/

- Normalization and standardization of the environment (within more mature teams), often by using a Shared Services teams, results in the consistent and reliable automation of controls (paragraph 2.3).

- The availability of automated controls within the Delivery pipeline opens the possibility for the application of a system-based audit based on Reperformance test procedures as much as possible instead of Procedural-based audits based on Inspection of samples and formalized documents (paragraph 3.3).

- For organizations which are not yet mature enough to apply higher levels of standardization and automated controls, the FEAT testing approach has been introduced to be applied until the criteria are met to use a system-based audit (paragraph 3.3).

- Use of the several available Culture frameworks and surveys to assess the maturity of DevOps within organizations and teams can help in properly tailoring the control framework (paragraph 3.2).

In paragraph 3.4 an updated maturity control framework has been presented which gives an overview of the controls that are necessary to be implemented within the Delivery pipeline in order to achieve the Change management control objective. Now organizations and IT auditors can also evaluate at which level of maturity they operate at mitigating risks in development and operations for their organizations.

Even if organizations have not yet adopted Agile and DevOps formally, it is recommended that audit, risk and security professionals keep these practices on their radar and develop an understanding of their characteristics. DevOps approaches might find their way into the organization rapidly (perhaps through shadow adoption or as the result of a merger or acquisition). ISACA emphasizes the importance of IT auditors being prepared and having a seat at the table, to be able to timely discuss the context, associated risks and relevant security and audit controls [37]. We all know Benjamin Franklin's famous quote "By failing to prepare, you are preparing to fail". IT auditors and other assurance professionals are in a symbiotic relationship with the IT department/teams and therefore must constantly prepare and gain knowledge of new IT technologies and principles.

Appendix A: Reference list

[1] Kim, G; Humble, J; Debois, P; Willis. (2016). The DevOps Handbook – How to create world-class agility, reliability, & security in technology Organizations. Portland, United States of America: IT Revolution press, LLC.

[2] Kim, G; Behr, K; Spafford, G. (2013). The Phoenix Project. A novel about IT, DevOps, and helping your business win. Portland, United States of America: IT Revolution press, LLC.

[3] State of DevOps 2018. Consulted on November 4th 2025 on https://live-puppet-p4.pantheonsite.io/resources/history-of-devops-reports#2018

[4] What is cloud.gov. Consulted on May 19th 2019, on https://docs.cloud.gov/

[5] How we work. Consulted on May 19th 2019, on https://18f.gsa.gov/how-we-work/

[6] IT Revolutions. (2015). DevOps Audit Defense Toolkit.

[7] Theory of constraints. Consulted on May 19th 2019, on https://nl.wikipedia.org/wiki/Theory_of_constraints

[8] Lean manufacturing. Consulted on May 19th 2019, on https://en.wikipedia.org/wiki/Lean_manufacturing

[9] Toyota Kata. Consulted on May 19th 2019, on https://en.wikipedia.org/wiki/Toyota_Kata#References

[10] Fowler, M; Highsmith, J. (2001). The agile manifesto. Consulted on May 19th 2019, on https://agilemanifesto.org/iso/en/principles.html

[11] Fowler, M. (2018 February 26th). The practical test pyramid. Consulted on May 19th 2019, on https://martinfowler.com/articles/practical-test-pyramid.html

[12] DORA DevOps Assessment. Consulted on May 19th 2019, on https://devops-research.com/assessment.html

[13] Microsoft DevOps self-assessment. Consulted on May 19th 2019, on https://www.devopsassessment.net/

[14] Toyota Kata. Consulted on May 19th 2019, on https://en.wikipedia.org/wiki/Toyota_Kata#References

[15] Secure Software Alliance – Framework Secure Software. Consulted on May 19th 2019, on https://securesoftwarealliance.org/framework-secure-software/

[16] Deming's principles. Consulted on July 11th 2019, on https://en.wikipedia.org/wiki/W._Edwards_Deming

[17] IT Revolution. (2018). Dear Auditor. DevOps community to Security with love. Consulted on July 12th 2019, on https://itrevolution.com/product/dear-auditor/

[18] https://en.wikipedia.org/wiki/Waterfall_model

[19] Benington, H.D. (1983). Production of Large Computer Programs. IEEE Educational Activities Department.

[20] Royce, W. (1970). Managing the development of large software systems. IEEE WESCON.

[21] Bell, T; Thayer, T.A. (1976). Software requirements: Are they really a problem? California. TRW Defense and Space Systems group.

[22] Singleton, T.W. (2010). The minimum IT controls to assess in a financial audit (part 2). ISACA journal volume 2, 2010.

[23] Casteren, W van. (2017). The waterfall model and agile methodologies: A comparison by project characteristics. Open Universiteit Nederland.

[24] DevOps Topologies. Consulted on May 19th 2019, on https://web.devopstopologies.com

[25] What is Infrastructure as code. Puppet labs. Consulted on May 19th 2019, on https://puppet.com/blog/what-is-infrastructure-as-code

[26] Fowler, M. (2006). Continuous Integration. Consulted on May 19th 2019, on https://martinfowler.com/articles/continuousIntegration.html

[28] Brodie, S. (2019). From Agile to DevOps to Continuous delivery. Consulted on May 19th 2019, on https://techbeacon.com/app-dev-testing/agile-devops-continuous-delivery-evolution-software-delivery

[29] AICPA. (2018). Understanding the entity and its environment and assessing the risks of material misstatement.

[30] Gartner. (2018). Seven Imperatives to adopt a CARTA approach.

[31] Lencioni, P. (2002). The five dysfunctions of a team. John Wily & Sons.

[32] PwC. (2013). Building digital trust – The confidence to take risks

[33] Costa, A; Anderson, N. (2010). Measuring trust in teams: Development and validation of a multifaceted measure of formative and reflective indicators of team trust. Brunel University, Uxbridge, UK.

[34] Westrum, R. (2004). A typology of organizational cultures. Quality Safety Health care publication 13(suppl 2).

[35] Google Project Aristotle. (2014): https://rework.withgoogle.com/intl/en/guides/understanding-team-effectiveness

[36] DevOps Enterprise Forum. (2015). Measure efficiency, effectiveness, and culture to optimize DevOps transformation. IT Revolution.

[37] ISACA. (2015). DevOps Overview. An ISACA DevOps series white paper.

[38] Plutora. (2019). Infrastructure as code: What is it, and why should my engineers care? Consulted on May 19th 2019, on https://www.puppet.com/blog/what-is-infrastructure-as-code

[39] Consultancy.nl. (2018). Continuous integration, continuous delivery: de stap na agile. Consulted on May 19th 2019, on https://www.consultancy.nl/nieuws/16755/continuous-integration-continuous-delivery-de-stap-na-agile.

[40] ISACA. (2018). COBIT 2019 framework. Governance and Management Objectives.

[41] Sondergaard, P. (2013). Everyone is a technology company. Gartner. Consulted on May 19th 2019, on https://blogs.gartner.com/peter-sondergaard/everyone-is-a-technology-company/

[42] Digital AI. DevSecOps Tools Periodic Table. https://digital.ai/learn/devsecops-periodic-table/

[43] Bagmar, A. (2012). Behavior Driven Testing (BDT) in Agile. Consulted on August 28th 2019, on https://www.slideshare.net/abagmar/anand-bagmar-behavior-driven-testing-bdt-in-agile

[44] Postma, S. (2015). Schuberg Philis Delivery Pipeline. Schuberg Philis.

[45] Fowler, M. (2014). Canary release. Consulted on August 30th 2019, on https://martinfowler.com/bliki/CanaryRelease.html

[46] Gangaram Panday, S. (2015). Introducing the Full population & Exception Analysis Testing (FEAT) method. Schuberg Philis.

Appendix B: Acronym list

AI	Artificial Intelligence
AICPA	American Institute of Public Accountants
API	Application Programming Interface
ASVS	Application Security Verification Standard
AWS	Amazon Web Services
BCP	Business Continuity Planning
BDT	Behavior Driven Testing
CAAT	Computer Assisted Auditing Technique
CAB	Change Advisory Board
CARTA	Continuous Adaptive Risk and Trust Assessment
CD	Continuous Delivery
CI	Continuous Integration
COBIT	Control Objectives for Information and related Technology
CD	Continuous Delivery
Dev	Development
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DORA	DevOps Research and Assessment
DRP	Disaster Recovery Planning
DSDM	Dynamic Systems Development Methodology (now Atern)
EDP	Electronic Data Processing
FEAT	Full population & Exception Analysis Testing
FedRAMP	Federal Risk and Authorization Management Program
FISMA	Federal Information Security Management Act
GRC	Governance, Risk and Compliance
IaC	Infrastructure as Code
IEEE	Institute of Electrical and Electronics Engineers
ISACA	International Systems Audit and Control Association
ISAE	International Standard for Assurance Engagements
IT	Information Technology
ITGC	IT General Controls
NOREA	Nederlandse Orde van Register EDP-Auditors
Ops	Operations
OS	Operating System
OWASP	Open Web Application Security Project
PaaS	Platform-as-a-Service
PRINCE2	Projects In Controlled Environments 2
RE	Register EDP-Auditor
RMM	Risk of Material Misstatement
SAS	Statements on Auditing Standards
SIT	System Integration Test(ing)
SSA	Secure Software Alliance
SSH	Secure SHell
ToC	Theory of Constraints
TRW	Thompson Ramo Wooldridge
UAT	User Acceptance Test(ing)
UK	United Kingdom
US	United States
UT	Unit Test(ing)
VCS	Version Control System
WESCON	Western Electronics Show and Convention
XP	eXtreme Programming

Appendix C: Initial waterfall phases

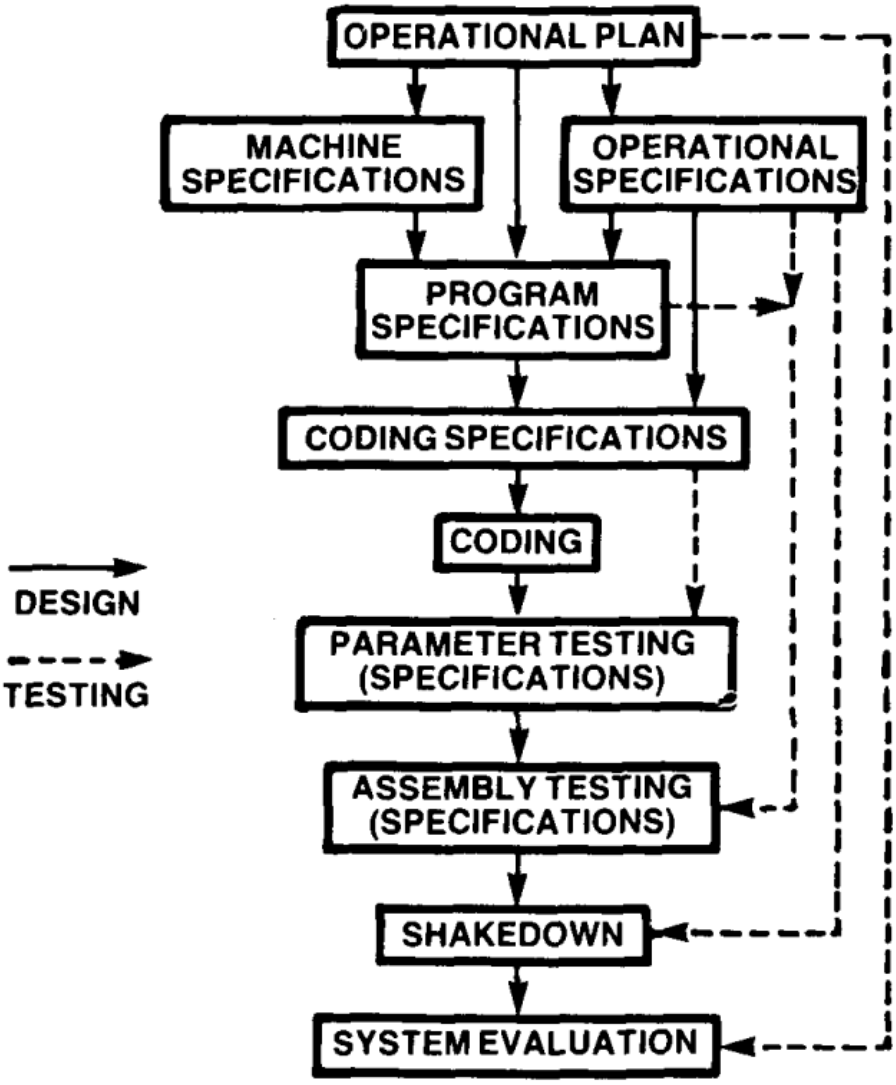

Figure 4. Program production. Production of a large-program system proceeds from a general operational plan through system evaluation; for example, assembly testing verifies operational and program specifications.

Figure 8: Initial waterfall phases [19]

Appendix D: Periodic table of DevOps tools

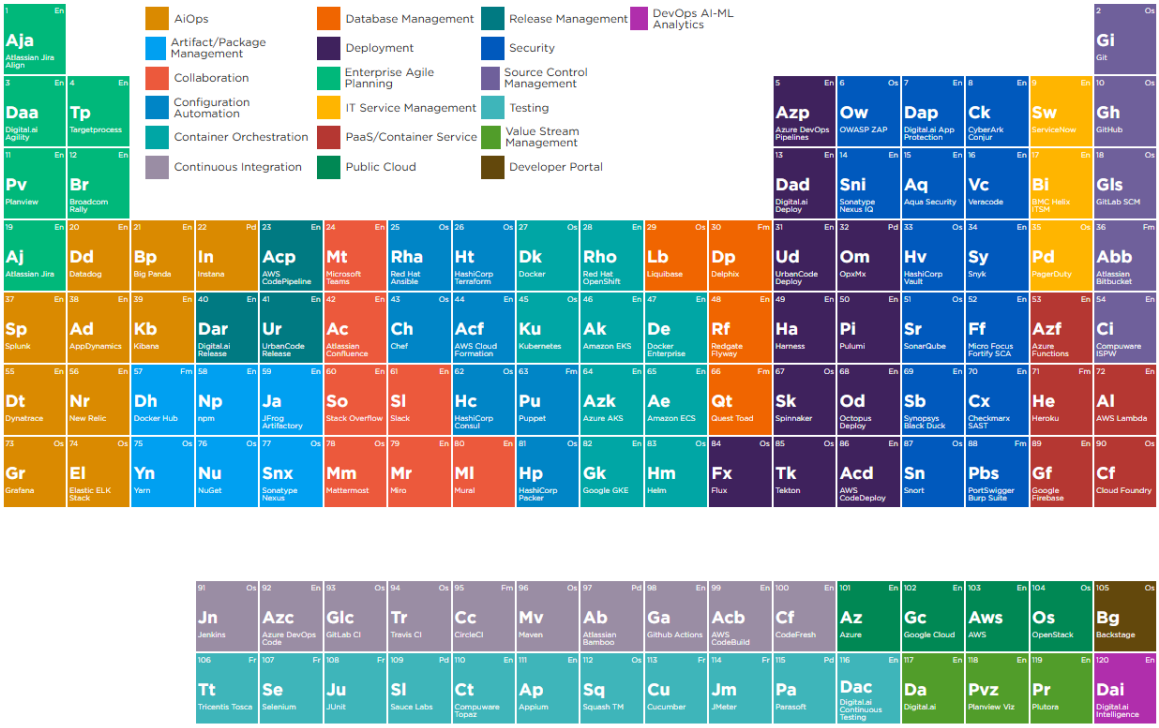

Figure 9: Digital AI. DevSecOps Tools Periodic Table [42]