NOREA
DE BEROEPSORGANISATIE VAN IT-AUDITORS

DevOps and Agile in control

A study report by NOREA

Author: S. Gangaram Panday MSc RE CISA – Schuberg Philis

PO box 7984, 1008 AD Amsterdam
phone: +3120-3010380
e-mail: norea@norea.nl
www.norea.nl

Contents

1. Introduction

'We moved to Agile ways of working about 10 months ago. Self-steered autonomous groups of engineers working in small teams, having full control over what tasks they put in the next sprint. Really, we are 100% Scrum nowadays. No more bureaucracy! No more paperwork and only limited documentation! Just full focus on delivering business features. As a matter of fact, we started automating large parts of our testing and integration steps, allowing us to release new features several times per day" – Typical IT manager, somewhere between 2009-2019.

And while the IT manager is explaining with full excitement how everything has changed, you as an IT auditor are stuck with the question of how this new way of working will fit into your existing control framework and beliefs regarding internal control? Self-steering? Limited documentation? Automated? Releasing continuously? …And who is keeping everything in control?

For sure you've already heard the terms Agile, Scrum and DevOps at some presentation or conference, but that initiatives like this have started in your organization, that is new. Don't these hypes fit better with innovative startups, or hipster tech-companies such as Google, Amazon and Netflix? Companies which are building a new future – greenfield –, rather than your company with all its legacy systems and pressure on compliance!

The fictional story above depicts a common reaction regarding Agile and DevOps by IT auditors. There is an underlying tone of disbelief or maybe even rejection, regarding this new way of working. A clear proof of this is the '*Dear Auditor. DevOps community to Security with Love*' letter written by the DevOps community in 2018 at the DevOps Enterprise Forum conference [17]. This letter is an attempt to improve collaboration and understanding between DevOps engineers and auditors.

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 newly 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. 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. In the end the 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 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].

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.

At the start of the writing of this 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 first 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 first iteration is by no means meant to be complete and covering the full picture but has a focus on the key risks and associated controls that must be managed effectively.

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 control framework is presented.

In chapter 4 the 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.

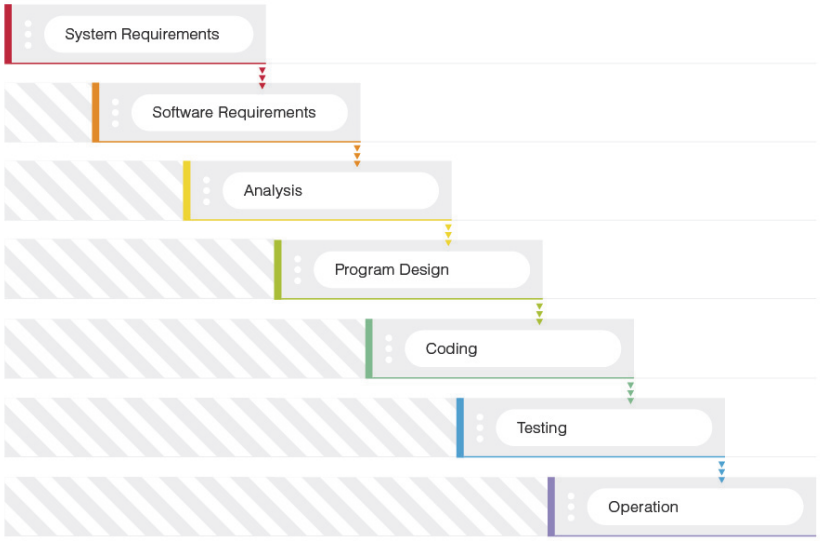

Figure 1: The waterfall model [23]

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 adaptive 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 Scrum, Extreme Programming, Crystal and Lean software development practices. In figure 2 below an overview is provided of Scrum for implementing agile principles.

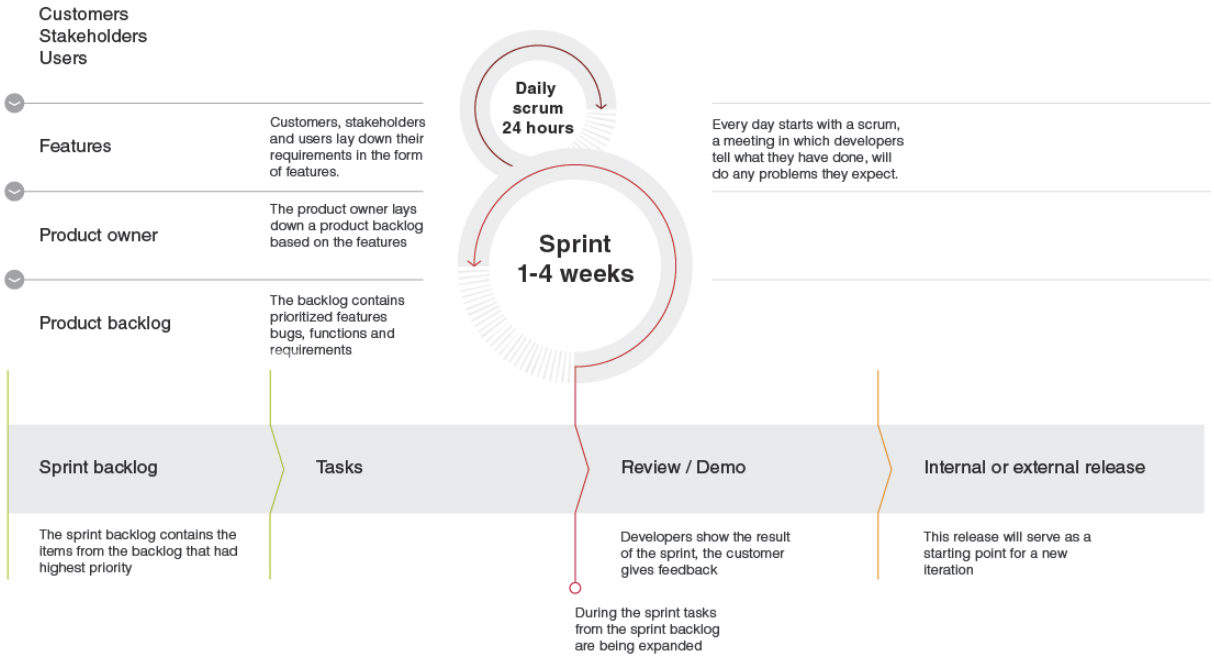

Figure 2 Scrum schematic overview [23]

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 (VCS) combined with automated builds (Continuous Integration). Both 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.

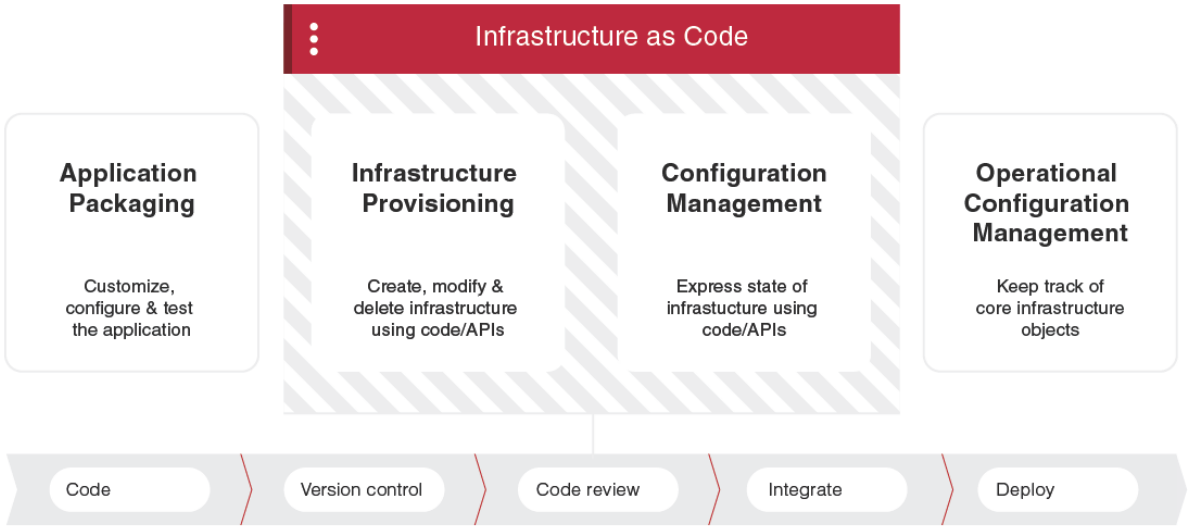

Figure 3 Infrastructure as code [38]

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 formal methodology, there exist many categories of DevOps implementation and almost all DevOps teams are unique implementations. The DevOps

[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.

Topologies organization presented 7 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 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.

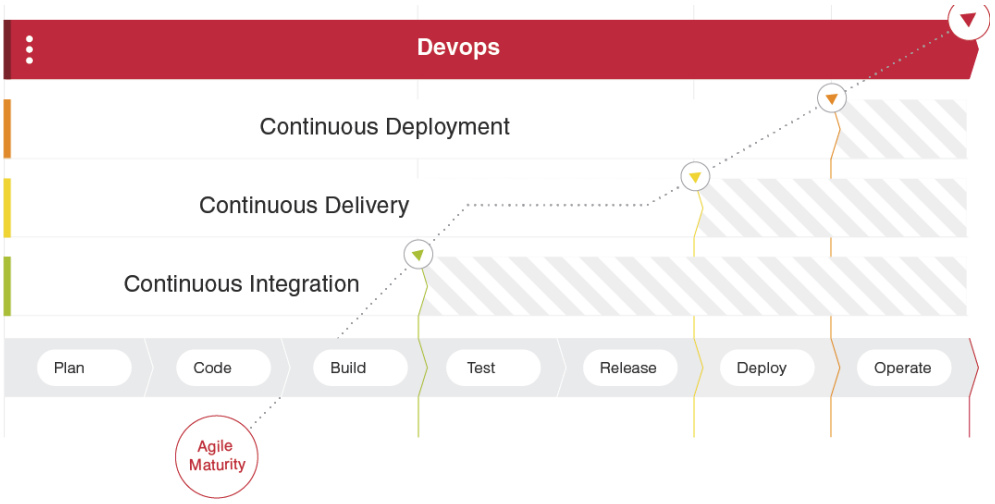

Figure 4: Comparison of CI /CD, Continuous Deployment

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 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, 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 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'.

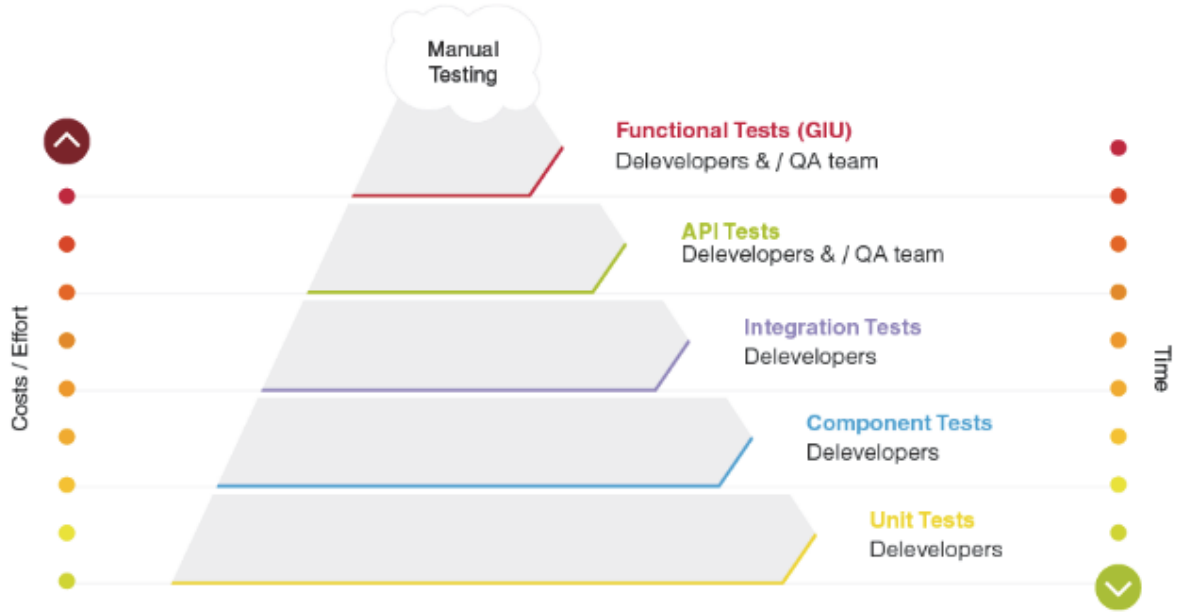

Figure 5 The ideal test pyramid [43]

3. Agile and 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.

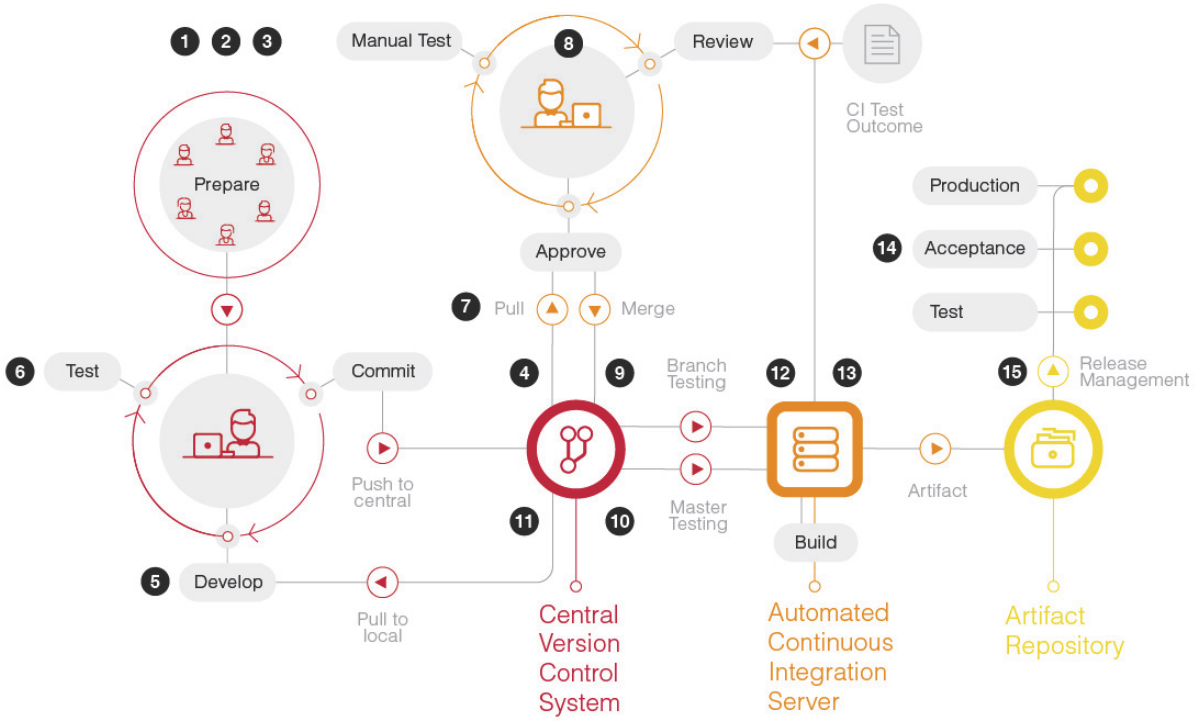

Figure 6: Example of a Delivery pipeline at Schuberg Philis [44]

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 in scope for the IT audit which are managed by DevOps teams, primarily the controls within the Change management domain will be affected, although other domains (e.g. Information security) might also be affected. The control objectives as such do not change, but the way the controls are implemented and therefore, how they need to be tested, is different. This has been confirmed by ISACA in their newly published COBIT 2019 framework (see paragraph 1.2). In the control framework presented in paragraph 3.4, the Agile and DevOps change management controls are presented. We believe that the auditor should assess these controls instead of the regular controls in the Change management domain.

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 DS9 Manage the configuration DS10 Manage problems	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
		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].

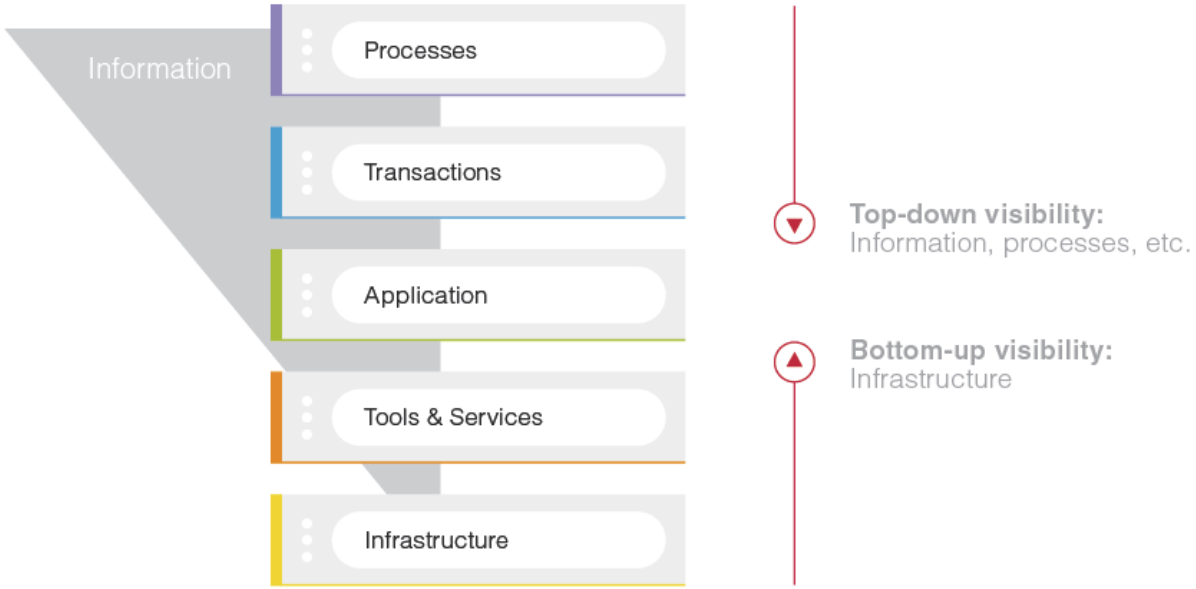

Figure 7: IT stack divided between DevOps and Shared Services teams [30]

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.

3.4. The control framework

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
1	Prepare	The team has chosen and documented an appropriate Agile methodology to facilitate their planning, requirements analysis and delivery process (e.g. Scrum, Extreme Programming, etc.). Based on the methodology chosen the proper roles are assigned and backlog items are defined.	1. In line with the company requirements and objectives, the development teams have selected and adopted a suitable development methodology which is properly documented. 2. Based on the selected Agile method, product owners are assigned and responsible for selected business lines and/or products. Feature requests are initiated by business stakeholders (customers) and reviewed by the product owner. Ideally, this is documented according to a Definition of Ready. If multiple product owners are involved due to cross business line involvement, the lead product owner takes the final decisions. 3. Ensure that all stakeholder requirements, including relevant acceptance criteria, are considered, captured, prioritized and recorded in a way that is understandable to all stakeholders, recognizing that the requirements may change and will become more detailed as they are being implemented. 4. The product owner reviews and adds "user stories" to the product backlog in line with the principles of the selected method. The backlog item is categorized, prioritized and Definition of Done is determined. The product owner continuously reviews prioritization of backlog items. For more guidance on Agile based planning, requirements and delivery we refer to the following example: https://www.scaledagileframework.com/	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	Partly
2	Prepare	Develop and document high-level designs for the solutions in terms of	1. Based on the service portfolio and the activities on which team members are working identify whether for the solutions in scope high-level designs are available. The level of detail maintained	BAI03.01 Design high-level solutions	Partly

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
		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. Ensure that stakeholders actively participate in the designs and approve each version.	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.		
3	Prepare	Procure solution components in accordance with requirements, detailed designs, architecture principles and the enterprise's overall procurement policies and procedures (considering security/privacy and compliance requirements).	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 and match this with the acquisitions in an asset inventory.	BAI03.04 Procure solution components	No
4	Develop	Maintain central control of software versions by using tools for enforcing automated version control on the code repository for	1. Validate if a central Version Control System (VCS) is in place and the team had enabled the proper configuration settings. 2. Validate at least if all code changes are logged (who, what, when) and if the log data is maintained for a sufficiently long period. 3. Validate if specific access rules are defined for the solution in scope and if those are properly implemented.	BAI03.03 Develop solution components	Yes

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
		Application, Infrastructure and Test code.	4. Validate that a branch policy is defined and followed (i.e., feature branches with review to enforce 4-eyes principle). 5. Validate if agreements are made that passwords, access keys and other sensitive information is not put into the code (in unencrypted format); ideally periodic scans should be run to uncover sensitive information in the code. 6. Ensure that the VCS tool is used as well for storing and maintaining the infrastructure code (unless not codified yet) and the test scripts.		
5	Develop	Develop solution components progressively in a separate environment, in accordance with company standards.	1. A coding standard is available. The coding standard contains guidelines for the use of (a) programming language(s), programming style, practices and methods. The code should be automatically tested for adherence to the coding standard. 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.). 3. Selection of external software components (including software libraries) is based on agreed upon guidelines to ensure compliance with security / maturity requirements.	BAI03.03 Develop solution components	Yes
6	Develop (test)	Testing of the changes made should be part of the development process. The developer should produce automated test cases (e.g., Unit Tests or Component Tests) that prove the code works as intended. The automated test cases are	1. A test approach is applicable requiring the execution of at least Unit Testing or Component Testing of the code changes made by the developer. 2. Validate if the test approach includes requirements on test coverage, in order to be able to continually evaluate the test effectiveness based on a required level of minimal test coverage (e.g., 70%) per specific code module.	BAI03.07 Prepare for solution testing BAI07.04 Establish a test environment BAI03.08 Execute solution testing	Yes

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
		included in the VCS together with the code, so that all future changes can be easily tested as well.	3. Validate that the developer updates any applicable automated acceptance tests when features are changed.		
7	Develop	A peer review of the code is mandatory for the code changes based on code review guidelines.	1. The team has a documented their code review guidelines for performing the peer-review e.g. based on best practices such as Google Style Guide or, based on the application context, enriched with security checks from the OWASP Application Security Verification Standard (level 1 through 3). 2. Once committed, the developer can push the local branch to the CVS. Ensure the developed code remains a branch in this stage, until further testing and merging/approval is completed. 3. The VCS enforces a peer review of the code change by another developer of the team who can pull the new code change for review.	BAI03.08 Execute solution testing BAI03.03 Develop solution components	Yes
8	Develop	A qualified peer reviewer performs proper testing including documentation of findings and merging of code takes place only after successful testing. All actions performed are logged in the VCS.	1. In case of findings, the peer reviewer provides comments to the original developer, who reviews and resolves these comments. 2. The merge action is only performed after successfully passing the peer review. If the peer review fails, the merge request is declined, and the developer is responsible to fix the identified shortcomings. 3. The peer reviewer checks automated builds and quality checks if included in the system (for example, SonarQube quality gates, unit test results, etc.) before completing the merge process (output of control #9). 4. The VCS is configured to log all merges to master including pull requester, merger, date of merging, reference to backlog or	BAI03.08 Execute solution testing BAI03.03 Develop solution components	Partly

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
			change ticket number, code change documentation and optionally the merger's assessment comments. 5. Peer reviews are performed by qualified developers that are knowledgeable with the existing code base.		
9	Build	During the build process both the infrastructure and application code is (automatically) tested and analyzed thoroughly on for example security vulnerabilities, dependencies, third-party libraries.	1. During the build process the following automated vulnerability scans should be performed (not all of it is common practice yet): 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. Validate the rules set by the team on failing the build (based on documented minimal requirements).	BAI03.08 Execute solution testing DSS05.02 Manage network and connectivity security DSS05.07 manage vulnerabilities and monitor the infrastructure for security-related events	Yes
10	Test	After the successful build the (automated) delivery process is started by commencing a set of tests to be run on the whole code base on production-like environments. The executed (automated or manual) tests checks the module on a code level (e.g. unit tests), if the component integrates	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 change. 2. Create a test environment that is commensurate with the enterprise environment (i.e., production-like). However, to comply with privacy laws and regulations appropriate rules should be 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).	BAI03.07 Prepare for solution testing BAI07.04 Establish a test environment BAI03.08 Execute solution testing	Partly

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
		successfully with the dependent components (e.g. integration tests) and if major features of the product work as specified (e.g. acceptance tests).	3. Ensure the 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.		
11	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. Document the minimum test coverage requirements per defined test and ensure that these are agreed upon with the product owner. 3. Set up test coverage monitoring and, if currently below minimum requirement, ensure that test coverage goes up over time (in line with agreements made with the product owner). 4. 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 issues are proposed towards the product owner for acceptance and proper tracking and management attention.	BAI03.08 Execute solution testing	Yes
12	Test	Tooling is used for performing integration testing on the created software builds, using predetermined test scenarios, to verify proper	1. Ensure that the test approach and test plan include (automated) integration testing to be performed on all merged changes. 2. Validate the integration test scripts for the inclusion of proper integration tests and use of proper tools for execution of the tests. 3. Acceptance of test findings that need to be resolved is discussed within the team. When resolution is not possible in the short term, acceptance of the test finding and moving to production	BAI03.08 Execute solution testing	Partly

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
		interoperation of (sub)systems.	without resolution has to be done by the product owner, where needed in consultation with affected stakeholders.		
13	Test	Based on the identified test approach, proper security scans on the finished code (static code test) are performed in a timely manner (e.g. vulnerability scanning, code dependency, penetration testing). Exceptions are documented, prioritized and followed up. (Not common practice yet.)	1. Ensure that the defined test approach and test plan includes security testing and specifies the applicable testing frequency, which should be based on the context and risk profile of the solution. 2. Ensure proper qualification of team members who perform or review security testing. 3. Ensure proper registration and prioritization of the test findings. Test findings are appropriately and timely communicated to the product owner and affected or involved stakeholders (e.g., when related to other domains such as privacy or compliance). Note: It is not mandatory that the security officer or the central security team is always involved in the process. However, periodic validation of the security test approach with experts from the central security team is considered best practice.	BAI03.08 Execute solution testing DSS05.02 Manage network and connectivity security DSS05.07 manage vulnerabilities and monitor the infrastructure for security-related events	Partly
14	Test	(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.	1. Create a test environment that is commensurate with the enterprise environment (i.e., production-like). However, to comply with privacy laws and regulations appropriate rules should be 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). 2. Ensure the test plan, set-up of the test environment and the test results are validated with the business stakeholder (product owner). 3. A register or log is maintained for test findings that need to be resolved. Tracking is performed in such a way that team	BAI07.03 Plan acceptance tests BAI07.04 Establish a test environment BAI07.05 Perform acceptance tests	Partly

#	Stage	Control description	Control assessment	Relevant COBIT 2019 controls	Automated control potential
			members can easily follow the resolution of these findings to ensure safe delivery. Note: In a modern CI/CD approach, where the automated tests in the Unit/Component Testing and Integration Testing phases cover all business rules, UAT tests are typically only needed to cover key usage scenarios.		
15	Production Deploy	Approved and tested deliveries are (automatically) deployed to the production environment.	1. Perform deployment based on the change management procedure describing the CI/CD process. The procedure should also describe the different change categories: Standard, normal and emergency changes. 2. Properly define the criteria for Standard changes (low-risk changes that are pre-approved e.g. infrastructure changes) and what the requirements for these changes are: 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 is available etc. 3. If possible, link the deployed changes to the respective change request tickets in the work planning tools (e.g. JIRA) to allow more context for the executed changes such as linking them to feature defects, incidents or user stories. E.g. by including the ticket numbers from the planning tools in the comments associated with version control check-ins which are linked to the production deployments. 4. Failed deliveries should have a clear fallback scenario (rollback / fix forward) and lead to a post-mortem review to analyze the reason for failure and optimize the delivery pipeline if possible.	Not applicable	Yes

4. Conclusion

DevOps has evolved from a niche practice into a mainstream strategy. State of DevOps, the largest DevOps survey report (30,000 respondents in 2018) reports an increase of respondents working in DevOps teams from 16% in 2014 to 29% in 2018, with respondents from 12+ industries with Technology and Financial Services being the largest reported industries where DevOps principles are applied [3].

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 when, who and what has changed (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).

- 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 a 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.

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, because 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 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 Franklins 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 May 19th 2019, on https://puppet.com/resources/whitepaper/state-of-devops-report

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

[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/forum-paper-downloads/.

[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://www.thoughtworks.com/continuous-integration

[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 organisational cultures. Quality Safety Health care publication 13(suppl 2).

[35] Google Project Aristotle. (2014): https://rework.withgoogle.com/print/guides/5721312655835136/

[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.plutora.com/blog/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] Xebia labs. Periodic table of DevOps tools (V3). https://xebialabs.com/periodic-table-of-devops-tools/

[43] Bagmar, A. (2012). Behaviour 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	Behaviour 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
CI	Continuous Integration
CD	Continuous Delivery
Dev	Development
DevOps	Development 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

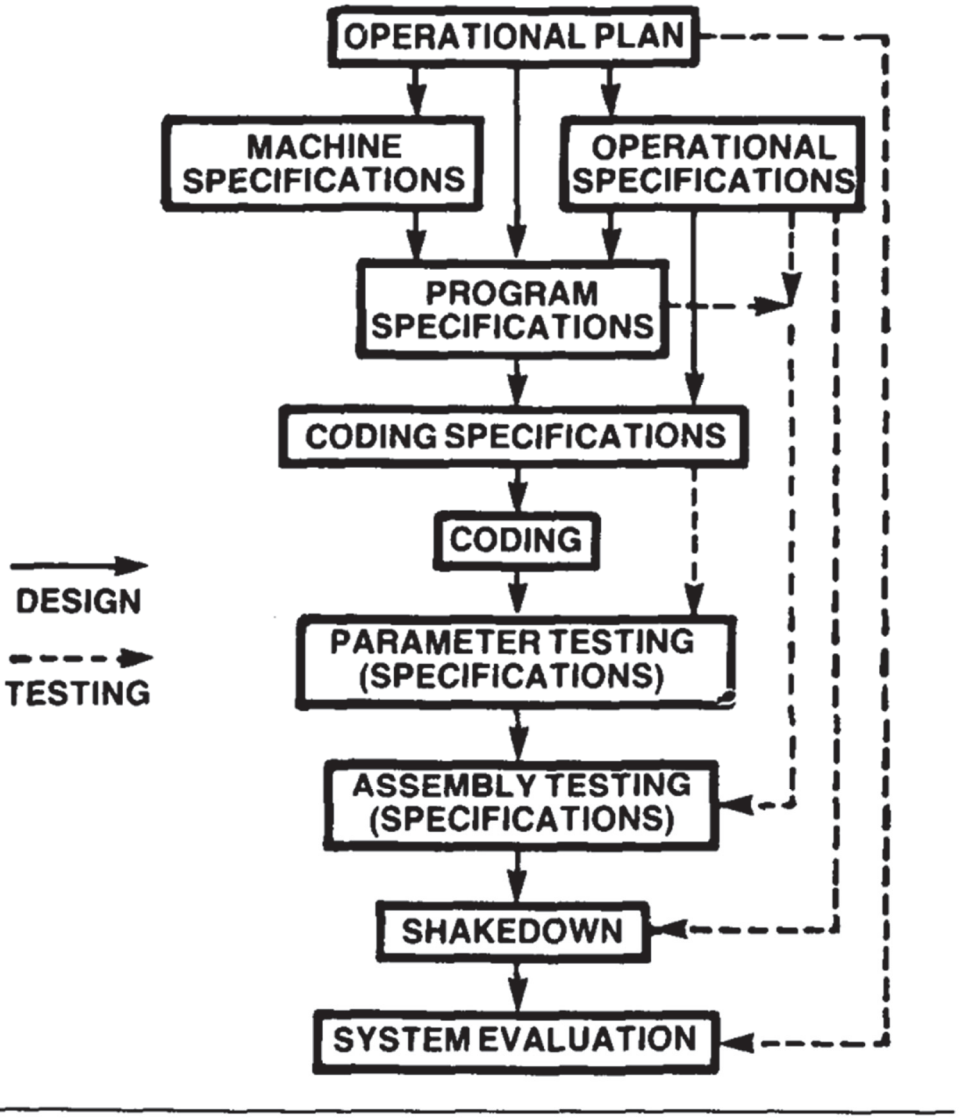

Figure 4. Program production. Production of a large-program system proceeds from a general operational plan through system evaluation; for example, assembly testing verifies operational and program specifications.

Figure 8: Initial waterfall phases [19]

Appendix D: Periodic table of DevOps tools

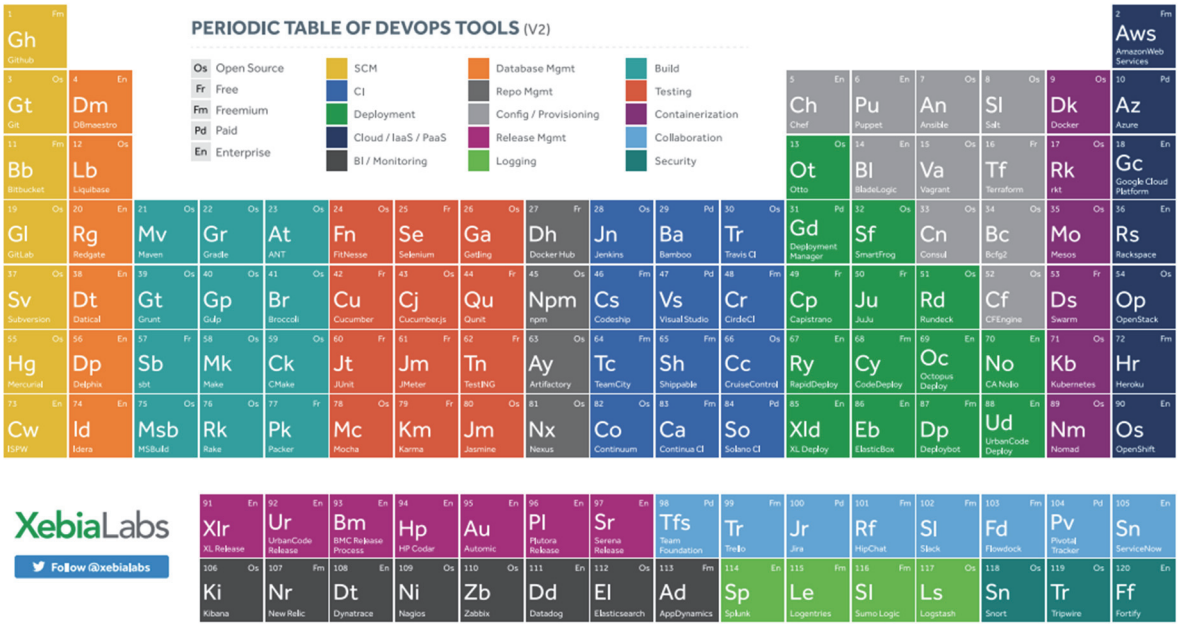

Figure 9: Xebia Labs Periodic table of DevOps [42]