Bringing Software Configuration Management and DevOps together

Lars Bendix[1,2], Christian Pendleton,[1,3] Erik Hochbergs[1,4], Laroy Nilsson Sjödahl[1,5]

Abstract:
With the current popularity of DevOps, many configuration management (CM) practitioners are faced with the problem of trying to figure out what kind of software configuration management (SCM) needs to be done and how it should be carried out in this, for them, new and unusual context of DevOps.

Configuration management is a product and methodology agnostic discipline. It can be used for software, hardware, cars, houses and trains – and it can be used in waterfall, agile – and DevOps. There exists a lot of literature on Software CM that explains what parts of CM to use when developing software and how to adapt and implement the abstract concepts and principles. Most literature is set in a waterfall-ish context, some of it in agile contexts – but so far there has been no help for SCM in DevOps contexts.

In this white paper, we will address the role of configuration management in a DevOps context. We identify the detailed characteristics of DevOps and discuss how they relate and map to the well-known SCM activities. We also provide some general guidelines for how to proceed with implementing a number of SCM activities on DevOps projects together with a number of practical examples that demonstrate the value of SCM activities in a DevOps context.

From our results in this white paper SCM practitioners will find help to identify and implement the right amount and type of configuration management in the right way for organizations using DevOps. It is our hope that also interested and motivated DevOps practitioners will find our white paper useful.

[1] Scandinavian Network of Excellence in Software Configuration Management, sneSCM.org
[2] Department of Computer Science, Lund University, Sweden.
[3] Curious Island AB, Malmö, Sweden.
[4] Axis Communications AB, Lund, Sweden.
[5] Eficode AB, Malmö, Sweden.

1. Introduction

Configuration Management is a fundamental part of any product development – and *Software* Configuration Management (SCM) of any *Software* product development. SCM provides the processes, techniques and tools that make it easier and safer for people to collaborate, enable different artefacts to float around in an organized and systematic way, and make sure that the right artefacts come together in a controlled way in a finished and coherent product. At the level of concepts and principles SCM is product and methodology agnostic and it is able to service and support any development method. This should also include "a DevOps context", which in this white paper means "a development organization that practices DevOps principles". So far standards and most books on SCM have been oriented towards supporting waterfall-like development methods, some things are known about how to carry out SCM for agile methods, but very little is known and available about what kind of SCM is needed and how the key concepts and principles could be implemented in a DevOps context. This lacuna is what we want to address in this white paper.

The cross-functional nature of DevOps means that each team – or even every person – should be able to take care of **all** aspects of software development – end-to-end. Everything from requirements engineering over coding and test to quality assurance, deployment and monitoring of the product in its production environment. An important consequence of this is that each team – or person – should also be able to take care of the necessary operational SCM activities and tasks. So, SCM is no longer something you can leave to the configuration manager specialist role. Furthermore, to be able to carry out the needed SCM activities and tasks, the right SCM competence needs to be present in the team – or in every person/developer.

Since very little knowledge exists about SCM in a DevOps context creates problems. Configuration managers who try to service and support a DevOps organization as if it was "business as usual" will probably have a hard time – if listened to at all. It might not be the usual things that will be helpful and useful for a DevOps organization – and for those concepts and principles that are valuable they will probably have to be implemented in different ways than on a waterfall project. DevOps practitioners might find themselves "ignorant" of long-established concepts, principles and techniques from the SCM world. That will cause them to struggle with problems for which they will eventually reinvent the wheel. Furthermore, they will not know what SCM concepts and principles might be behind some DevOps techniques and practices and thus in danger of making mistakes when they try to adapt a DevOps practice or technique to their specific context. To be able to help these people struggle less, we first have to establish how different DevOps characteristics relate to SCM activities, if all DevOps characteristics have SCM support, and if all SCM activities are needed. Then we have to give some general guidelines and examples for implementing relevant SCM concepts and principles in a DevOps context.

The primary target group of this white paper are *configuration managers* who are confused about how to act and what to do to fit in and be valuable and helpful also in a DevOps context. So, we assume that you have a lot of practical SCM experience from other contexts, but also that you have some theoretical knowledge about SCM concepts and principles. Furthermore, we hope that the white paper can also be helpful to *interested and motivated DevOps practitioners*. If you do not have access to a configuration management expert who can explain SCM for you, you could start with chapters 2.4 and 4.1-2 from [HS, 2020] and select further reading from their reference list.

This white paper is in large parts based on the master's thesis of Hochbergs and Sjödahl [HS, 2020]. However, their master's thesis has a broader scope and target group – and has a lot more pages that allow them to go deeper into details on certain aspects.

In the following, we will first sketch what we find more specifically characterizes DevOps. We need to do that both to define what *we* mean by DevOps in *this* white paper and to serve as the foundation for the next chapter. In that, we dig deeper into how the different DevOps characteristics relate to the well-known SCM activities and vice versa. This allows us to identify what SCM activities could be potentially helpful if there are problems with a certain DevOps characteristic – and what DevOps characteristics would suffer if we left out or ignored a certain SCM activity. Finally, we sketch some guidelines for how to proceed and examples of how important parts of SCM can be implemented in a DevOps context.

2. Characteristics of DevOps

To create a mapping between DevOps and SCM, we first need to know more about the characteristics of DevOps and the activities of SCM. For SCM activities it is easy since SCM has well established definitions and terminology (see chapter 3), but for DevOps it is slightly more complicated. There seems to be many opinions about what DevOps is and there seems to be no common agreement on a universal definition of the term DevOps [Wikipedia, 2021]. We do **not** want to enter into that discussion, since a precise and formal definition of DevOps is not what we need for this white paper. We need something more detailed and operational. We need to know what are the specific and detailed characteristics of DevOps, so we can capture the varied DevOps realities that must be supported from an SCM point of view.

We do **not** want to convince you of the advantages of DevOps – others will have to do that. Nor do we try to teach you DevOps or explain how to introduce it in your organization. We just want to make it absolutely clear and explicit what **we** mean in *this white paper* when we talk about DevOps and do that at a detailed level that makes a mapping between SCM and DevOps rich and useful. In a certain sense we are looking for the union of things that various people consider to be specific characteristics of DevOps – and for which SCM has to provide service and support – to make sure that we cover all aspects. If you don't agree with our results, it should be fairly straightforward to adjust our results from chapters 2, 3 and 4 to match your ideas of what DevOps is.

In the original work [HS, 2020], the investigation of DevOps characteristics was carried out both through a literature study and through interviews with DevOps practitioners. We build on that work and extend it with personal experience and further literature studies. If you want to know more in general and specifics about DevOps, you could read the relevant parts of [HS, 2020] (chapters 2.3 and 3 plus their reference list) and the current description of DevOps at Wikipedia [Wikipedia, 2021] and some of its references. For a deeper knowledge dive in the area, we recommend reading of [Forsgren et al., 2018] and for more data driven understanding of the effects of a DevOps transformation the various State of DevOps reports – in particular the 2018 report from Puppet [Mann et al., 2018] which mentions the importance of software configuration management without going into any detail.

Kent Beck has built the practices in eXtreme Programming on a number of general principles that are in turn built on five fundamental values [BA, 2005]. We will pursue a similar approach and first address the purpose and philosophy of DevOps before diving into the more specific characteristics. What is the purpose of DevOps? Why do we want to do it?

If DevOps is the solution – what was then the problem? In our opinion, DevOps addressed – or at least does so today – a number of different challenges in modern software development.

First, from the very word DevOps, it tries to calm the organizational divide (or silos) between Dev (who live by and get paid for change) and Ops (whose mission is to maintain stability). In one silo Dev do not understand why their changes cannot be put into production immediately so they can get closure – and in the other silo Ops do not understand why Dev continuously want to change something that works and is stable. The idea is to physically place Dev and Ops together such that they can communicate directly and get a better understanding of each other's challenges. Breaking down silos has been taken one step further, so it is not just silos between Dev and Ops, but also all other silos in the organization, like between requirements and coding, or between coding and test, and so on. The motivation

for this is the realization that handovers from one silo to another introduce slowness in the development process. We can get rid of the handovers by removing the silos and create cross-functional teams – or people – that can handle development end-to-end.

Another problem is heavily user-centric development where the development organization can not speak directly with the users to understand what they want. Like Spotify, Google, Atlassian, ordering food or a cab, or renting an electric scooter. Situations where development is very driven by what users need, but where we cannot directly ask them what it is that they are interested in. And where there is a very competitive market that will discover and satisfy user needs before us if we are not faster. In such a situation DevOps is a way of entering into a dialogue with users by proposing something and then observe and measure how the users react. This means that there is a very strong focus in DevOps on getting feedback from users about the things we propose [Abildskov, 2020]. However, more in general it has turned into providing meaningful feedback as fast as possible to all actors involved in software development. Project managers want to have fast feedback on project status, product owners want to have fast feedback on quality and feature completion, developers want to have fast feedback on builds, test results and bugs.

A key focus in DevOps is that of speed. In part driven by a very competitive market, in part by impatient actors, who want to have their feedback as fast as possible. We want the time from "business idea" to "user feedback" or from "committing code" to "code in production or rejected" to be as short as possible. There are a number of consequences of "fast feedback". In order to limit the time "from idea to production" we need to work with small increments and in small batches. Ideally, we would like to have the batch size of changes travelling together equal to one single, logical change. This means that one change does not have to wait for other changes to finish before it can be put into production. This also aligns very nicely with the business goal that a feature that can create revenue should not sit waiting for the next release but be put into production as quickly as possible.

With a batch size of one single change, we will have to repeat certain things (building, testing, QA, deployment, etc) over and over again, which makes manual processes a bottleneck and leads to a focus on automating as much as possible. When we deploy only once every six months there is very little motivation for and need to automate a (partly) manual 4-hour deployment process – that changes drastically when we have to deploy 10 times a day. With increments being very small there will be many increments and together with the need for speed it means that we will no longer have time for the traditional handovers between e. g. a requirements engineer and a coder, or a coder and a tester. Again, people – or teams – will have to be cross-functional.

DevOps also puts an extreme focus on quality. To be able to continuously deploy to production, we have to be convinced that the code quality is continuously high enough for release. If we don't maintain this level of quality, we will not be able to focus on speed either.

The above reflections can be summed up by the following quote stating that DevOps is "a set of practices intended to reduce the time between committing a change to a system and the change being placed in normal production, while ensuring high quality" [Bass et al, 2015].

These were some of the overall ideas and principles behind DevOps. With this first foundation for DevOps, we can now move on to uncover its practices. We will identify them

as more detailed and specific characteristics that are important for this white paper since we will need them in the following to create a useful mapping between SCM and DevOps.

The original work [HS, 2020] identified a number of more detailed characteristics of DevOps. They are all represented in the graph in figure 2.1 (taken from [HS, 2020]), which also shows the relations between the characteristics.

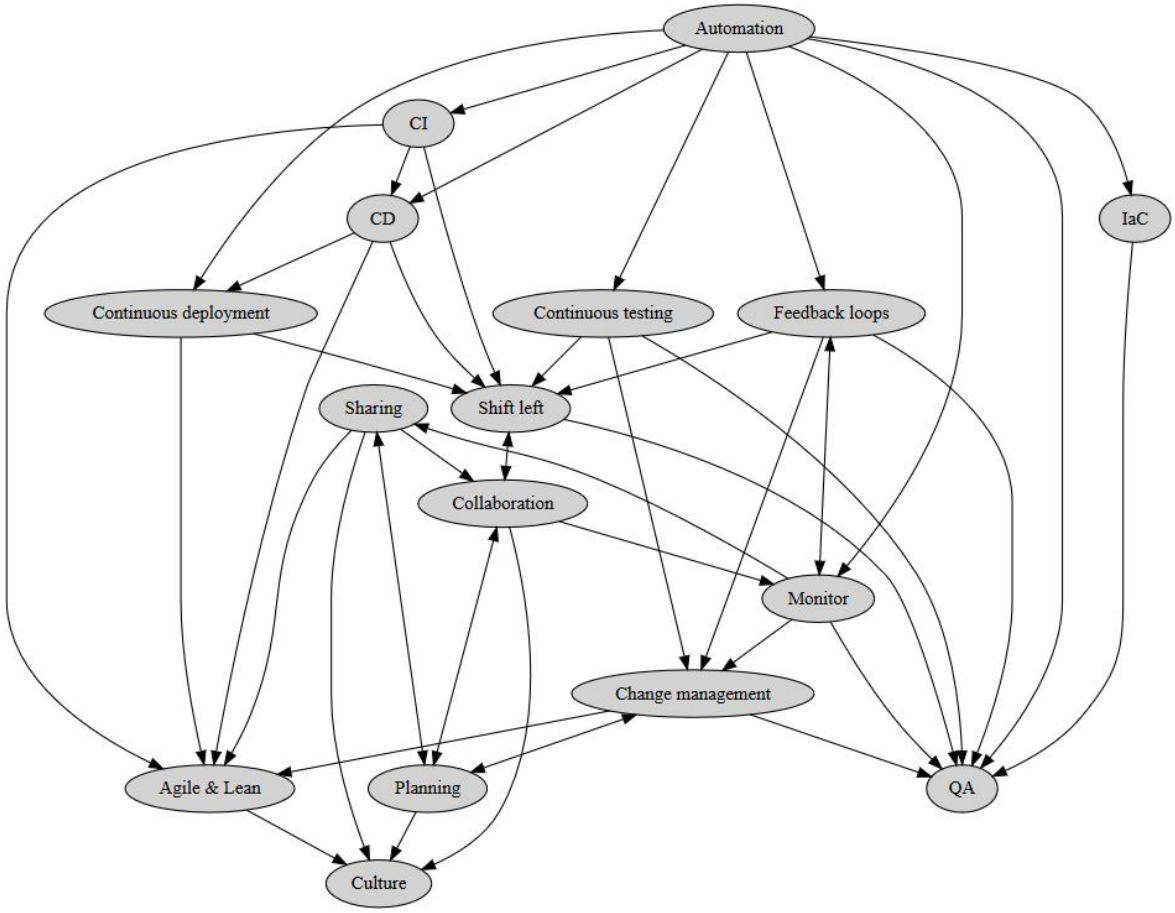

Figure 2.1 DevOps characteristics and their relations (from [HS, 2020]).

In the above graph, the nodes represent the identified characteristics of DevOps and the relations between characteristics are represented by directed arrows. The characteristic at the arrowhead is supported by the characteristic at the arrow's origin. Some relations are bi-directional meaning that the two characteristics mutually support each other. The node "IaC" stands for "Infrastructure as code", the node "CI" stands for "Continuous integration" and the node "CD" stands for "Continuous delivery" to distinguish this characteristic from "Continuous deployment". For a more detailed and complete description of and motivation for the identified DevOps characteristics and their relations, you should consult the original work [HS, 2020].

Care should be exercised not to over interpret this graph – in particular in the context of this white paper. We have decided to show the whole graph from the original work without any changes. This both to provide a richer picture for the interested reader than strictly necessary for this white paper and because we want to encourage people who are particularly interested in digging deeper into the DevOps part to consult the original work. In this white paper, we

focus entirely on the nodes to find the DevOps characteristics that we want to map to SCM activities.

Given that we use the whole graph and someone may be curious about details in the graph a little further explanation might be needed here.

When studying the graph, it is important to understand that the relative position of the nodes has no semantic meaning. There is no top-down structure intended. So, “Automation” does not precede “QA” or “Feedback loops”, neither is it on a higher level or more important than the two. In fact, you could rotate or twist the graph any way you want and have any characteristic “come out on top”. Likewise, quantity of relations is not a sign of importance. So, many input arrows (like for QA) does not mean that it is a more important characteristic than others – just that this characteristic gets support (maybe only in part) from many other characteristics to be carried out. Similarly, many output arrows (like for Automation) does not mean that it is a more important characteristic than others – simply that this characteristic (or parts of it) contributes to carrying out many other characteristics.

We are aware that this graph is neither perfect nor complete. You could argue for additional relations (like from Continuous testing to CD) or additional characteristics (like splitting up Agile and Lean). However, we see it as an important first step in establishing the more detailed characteristics of DevOps we need for our work on how SCM relates to DevOps and for the purpose of this white paper we find it a sufficient help in its present state. We are also confident in the solidity for the identified characteristics where most of the work was focused.

If you are interested you can further develop the graph, especially for the types of relations which are here only focused on “supported by” relations. In the original work, a lot of effort was spent on the relations between the DevOps nodes. An early version of the graph contained more relations and types of relations, but it created too much clutter in one figure.

So, the result of this chapter is that *we* consider DevOps to be characterized by the practices represented by the nodes in figure 2.1. In the next chapter, we will use these characteristics together with the SCM activities to create a detailed mapping between DevOps and SCM.

3. SCM in DevOps

We are now ready to create the sought-after mapping between Software Configuration Management and DevOps. This mapping will make it possible to address how SCM and DevOps are connected and related. More importantly it will help provide answers to the questions about what kind of SCM needs to be done and how it should be carried out in a DevOps context.

In order for SCM to become "operational" in a DevOps context we need better and more fine-grained relations between the two than just "SCM is necessary" or "as SCM we can service and support any development method". You might know about SCM, but not know what parts to pull out in certain situations or where a certain part of SCM could match a DevOps problem. Or you might know about DevOps, but not much about SCM and thus not know where to look for more specific help from SCM.

Building on the identified detailed DevOps characteristics from the previous chapter, we now need to identify what are the detailed SCM activities before we can start analysing more in detail the possible relations and connections between the two that will make up our mapping model. We will first present and motivate the model and then show how the model can be used in general to answer the overall questions. In the next chapter, we will provide more detailed and concrete examples of practical use of the model.

3.1 Mapping SCM and DevOps – the model

The original intention with the model (figure 3.1 below) was to come up with a tool that its creators could use to better explore SCM in DevOps context [HS, 2020]. Ultimately it would help them in answering their academic research questions in a systematic and structured way by relating each SCM activity to a subset of DevOps characteristics and vice versa. However, it turns out that the model can also be an excellent tool for SCM and DevOps practitioners. The research questions, which are also of interest to both SCM and DevOps practitioners, were:

- What SCM concepts and principles are needed and not needed in a DevOps context?
- What new SCM concepts and principles must be added to the SCM toolbox – if any?
- How to adapt relevant SCM concepts and principles to a DevOps context?

The first two research questions we will deal with in section 3.2, while we will treat the third one more in detail in chapter 4. But first we will introduce and motivate the model itself.

The starting point was a list of detailed DevOps characteristics on one side and a list of detailed SCM activities on the other side – followed by a careful analysis of how they can be related to each other. **The detailed DevOps characteristics** for this white paper were identified in chapter 2.

The detailed activities of SCM are well-known and documented in literature and were confirmed by [HS, 2020] through a series of interviews with SCM practitioners. For this white paper, we assume that you are acquainted with the SCM activities (shown on the left side in figure 3.1) – if not there are plenty of places where they are defined and described. For one, sections 2.4 and 4.1 plus the reference list from the original work [HS, 2020] will serve as an excellent starting point. Just like you did not need to have a good understanding of

DevOps to understand the figure in the previous chapter, then you don't need to have a deep understanding of SCM (or DevOps) to understand the figure in this chapter. Obviously the better you understand one or the other or both, the more you will get from the model.

The relations and connections between SCM activities and DevOps characteristics were established by a careful analysis of SCM activities and DevOps characteristics. The type of relationship that was sought was that of an SCM activity *supporting* (in part) one or more DevOps characteristics – or a DevOps characteristic being *supported by* (in part) one or more SCM activities. In each case the analysis also took into consideration the internal connections between SCM activities and/or DevOps characteristics to provide as much information as possible for the analysis. In order to make the model as clear and simple as possible, the internal edges within SCM activities and DevOps characteristics have been omitted, but keep in mind that these relations still exist and were used for the analysis.

The complete analysis resulted in the model seen in figure 3.1 below.

We decided to use the figure as is from the original work [HS, 2020] except from cropping the right part which related the detailed DevOps characteristics further on to more general DevOps activities. We did that because we liked the simplicity of the figure and found it useful and expressive and thus saw no reason to "draw it again". However, the cropped part is out of scope for this white paper as we only focus on the relations between SCM activities and DevOps characteristics, so we had to leave that out.

We acknowledge that the model is not perfect – and probably never will be. In part because you can never have enough data to perfect the model, but most of all because the model is a generic outcome of interviews with several development organizations.

There may be missing relations ("IaC" could for example have a relation to "Version Control") and there may be relations that are not that strong and maybe should not be there. Likewise, leaving out the internal connections might turn out to be an oversimplification and adding them might be worth the extra complexity. Relations and connections in the model were also the results of input from a number of companies and specific contexts. A lot of discussion and evaluation resulted in the present model as a generic representation that captures the most important and general aspects but will also never be able to match all contexts out of the box.

However, you should be able to use the model as a starting point and tailor or further develop it to match more closely your specific company or situation and base it on your actual use of SCM and DevOps. Some activities or characteristics could be grouped, some relations could be coloured to highlight what is important and in focus for you right now. You could also remove relations that are not important to you in the present moment (e.g. SCM plan) to get less clutter and more clarity.

In the agile, iterative spirit of DevOps, we consider the presented model to be an excellent starting point. Time and more experience with using the model in practice will show if and how this generic model can be made more complete and correct without adding too much clutter and complexity.

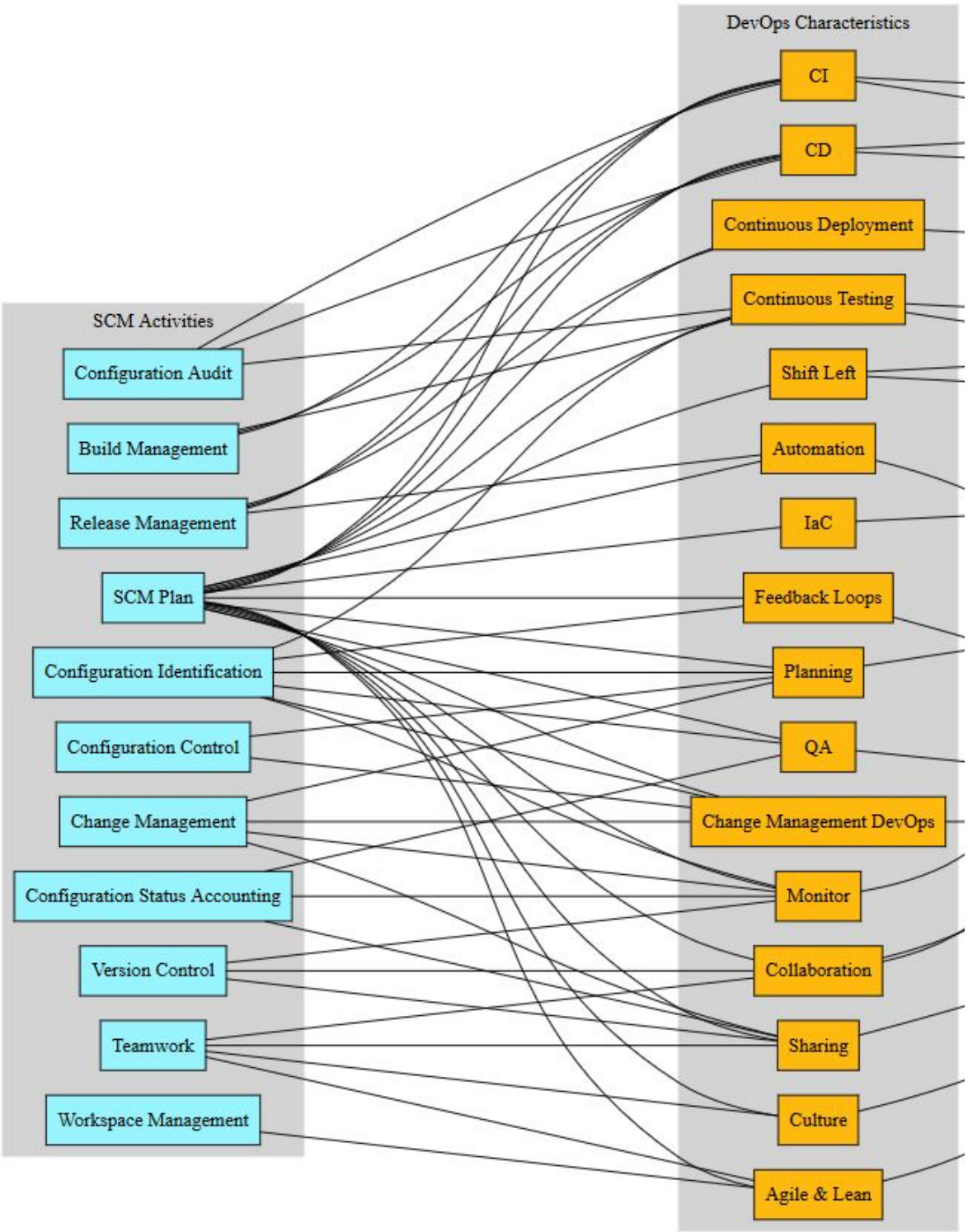

Figure 3.1 Connections between SCM activities and DevOps characteristics (adapted from [HS, 2020]).

3.2 SCM's role in a DevOps context

Now that the model is in place, we can start exploring and exploiting it. First, we will use the model to answer the research questions from the previous section. These research questions are not only of academic interest but are also very useful for practitioners. Then we will show some general examples of how both SCM and DevOps practitioners can use the results. We will leave more specific and detailed examples to the next chapter.

The primary purpose of creating the model was to be able to find and substantiate answers to research questions of an academic research project. However, since these questions are also highly relevant in an industrial setting, we include their answers in this white paper [HS, 2020].

The first research question was: "What SCM concepts and principles are needed and not needed in a DevOps context?" It is evident from the model that all SCM activities are connected to at least one DevOps characteristic and that most SCM activities have two or more relations to some DevOps characteristic. So, the overall conclusion is that *all SCM activities are needed in a DevOps context*. The second research question was: "What new SCM concepts and principles must be added to the SCM toolbox – if any?" From the model it is clear that no DevOps characteristic is left without a relation to at least one SCM activity, this is a strong indication that SCM can provide some kind of help and support for all aspects of DevOps. On the other hand, it cannot be ruled out that new additions to the SCM toolbox could provide even better support in a DevOps context, though we see that more as implementing already existing concepts and principles in new and different ways as outlined in chapter 4. So, the overall conclusion is that *no new SCM concepts or principles are needed in a DevOps context*. However, SCM alone is not supposed to solve all problems in DevOps.

In summary, the SCM experts can remain confident that they do not have to learn new SCM activities – and that all of their SCM knowledge will also be useful and valuable in a DevOps context. The DevOps practitioners, on the other hand, will be relieved to learn that for (parts of) all DevOps characteristics there will be good help and support to get from SCM. Though the concepts, principles and activities of SCM will remain invariant and useful in all contexts, the actual way in which they are implemented will be different from context to context as we will show in chapter 4.

The activity "*SCM plan*" sticks out a little, being the only SCM activity that has relations to all DevOps characteristics. "SCM plan" is the part of the well-established SCM activities that keeps all the more detailed SCM activities tied together and shows how they depend on and relate to each other. The "SCM plan" for a specific product will ensure that all work on the product is handled and managed in a consistent way no matter who works on the product and when. This will be no different in a DevOps context. However, some of the principles in DevOps mean that we have to approach the *implementation* of the "SCM plan" activity in a completely different way in a DevOps context.

For instance, will the principles of speed and small batch sizes mean that we could have several deploys per day. As a consequence, there will no more be the time nor people to do the usual two-week Configuration Audit before our bi-annual release. The increase in the number of deploys will cause an increase in the operational SCM work – and the speed will mean that work has to be done faster. The cross-functional nature will mean that there will be less and different handovers for SCM to take care of – and since the DevOps practitioners

take care of all tasks and activities end-to-end there will be no need for operational SCM specialists. So, it seems like SCM work becomes more of a strategic nature in a DevOps context and that (most of) the operational SCM work will be carried out by DevOps practitioners – or tools like for IaC.

One thing that changes when the operational SCM work is (mostly) carried out by DevOps practitioners is *the primary target group for the SCM plan* – it changes from being SCM experts to being DevOps practitioners. This means that the SCM plan must go into more detail since their SCM knowledge will probably not be sufficiently deep to allow them to provide precise implementations to more general and generic descriptions. We will have to abandon the idea of a big, generic SCM plan up front in favour of iteratively and incrementally developing an SCM plan that consists of a number of specific SCM process descriptions that will exactly match the needs and culture of the product and organization at hand. This is very much in line with the agile spirit of DevOps and should guarantee that the SCM process descriptions are always updated to reflect what is actually being done.

This will also avoid the risk that the SCM activities become separated from the DevOps activities instead of seamlessly integrating them and letting the SCM activities be an integrated part of the workflow. There will be separate documents describing parts of the development process, practices and checklists. Collectively all these documents will make up the "Process plan" for a specific product of which the "SCM plan" will be an important part. It is vital that all the documents are shared so people can access them whenever they need – and that they are open for change so they remain living documents and can be kept consistent with the carried-out processes whenever they change.

In summary, it looks like most or all of the operational SCM work in a DevOps context will be carried out by cross-functional DevOps practitioners and leave less – and different – work for SCM practitioners. This work will be predominantly strategic SCM work. Working with the structure and generic contents of the SCM processes, educating the people responsible for the operational parts of SCM and continuously working on *SCM planning* rather than a static SCM plan up front.

Leaving behind the academic interest, how can the model be useful to practitioners – SCM and DevOps? The beauty of the model is that since relations work both ways the model will be valuable for both groups. You can start from a specific SCM activity and trace that to all the relevant DevOps characteristics where this activity can be helpful – or you can start from a specific DevOps characteristic and trace that to the relevant SCM activities that can support this particular characteristic.

If you are an SCM practitioner, you can use the model to focus the search for proper SCM concepts and principles in your SCM toolbox, when you are asked for help with a certain DevOps characteristic. If you have particular expertise with some aspects of SCM, the model will tell you for which parts of DevOps your expertise will be valuable. So, if you have deep and rich expertise on "Version control", you will be a valuable resource for strategic help with "Monitor", "Collaboration" and "Sharing" in DevOps.

If you are a DevOps practitioner, you can use the model to ask or look for help in a more specific and precise way. If you are experiencing problems in a certain aspect, you can figure out what DevOps characteristics are relevant and use that "terminology" when you communicate with the SCM expert. In case there is not an SCM expert available, you can use

it to know what parts of SCM theory you have to selectively read up on. So, if you experience problems with “Collaboration” in DevOps, you only need to read up on the “Version control” and “Teamwork” activities in SCM theory.

Finally, you could consider the model to be a “cheat sheet” giving you a good overview of the most important things – then you can (in other places) dig deeper into what is most valuable or important for you right now. Both for “studying” SCM, and for “studying” DevOps.

In the next chapter, we will dig deeper into some of the detailed answers to the third research question: “How to adapt relevant SCM concepts and principles in a DevOps context?”

4. How to proceed

Now that we have created a model that captures the relations and connections there are between SCM activities and DevOps characteristics, we want to show you more in detail how this model can be put to practical use.

The model in chapter 3 is not perfect, but it is a good source for what SCM areas to consider reshaping when the organisation is striving for a DevOps oriented way of working. We will dig deeper into how SCM activities can be used to support different DevOps areas and what aspects of the SCM activities become essential in various parts of DevOps. This way we will add an operational aspect to the model.

First, we will give some more detailed examples from the SCM point of view, then we will explain how things look like seen from the DevOps point of view.

4.1 SCM activities giving support to DevOps characteristics

Let us put on our configuration manager glasses and look at how some of the SCM activities can support DevOps characteristics. We will explain the connection and why the SCM support is important and what the configuration manager expert needs to focus on to give the best support.

4.1.1 Configuration Status Accounting – *supports: QA, monitor, sharing*
Quality assurance without status accounting doesn't make much sense. We need to get status information both when our product is being tested and when it is running in production. Configuration status accounting will here support gathering of status information, storing it in an identifiable and traceable way and eventually also display it in dashboards, reports etc. We can for example display values like MTBF (Mean Time Between Failure) and MTTR (Mean Time To Recover) for both test environments and production environments continuously in dashboards to make it easy to see emergency situations, trends etc for the product. Configuration Status Accounting can also gather data of what parts of the product have a higher frequency of errors and/or faulty behaviours. This data can be visualised to the development teams and be used as input for backlog prioritisation.

The DevOps area Monitoring is feeding the status accounting area with data from observing the product in various ways. The SCM work with status accounting turns this data into information, visualised in a proper way for developers, product owners, coordinators and managers of various kinds and makes it possible for the organisation to digest and use the information. For example, by using monitoring to gather data about feature usage in a product and then visualise this to product management, we can give valuable input to innovation and prioritisation of future work. It can also make it possible to determine the business value of implemented features.

Transparency and information sharing are important aspects of DevOps and we can use activities from SCM to support this and make it easier to tear down the silos in the organisation. When we share information, it is important that we can rely on the accuracy, trace the information back to its origin data and that we can connect the information to the right product configuration. Working routines for configuration identification of all kinds of information is of help here, particularly version identification (numbering) of configuration

items that we want to share. We can also use baselines to create various configurations of items to ensure that the information is received in its correct context.

4.1.2 Configuration identification – *supports: continuous testing, feedback loops, QA, monitor*

Continuous testing and QA both rely on the fact that we can handle test data and test environments as configuration items with proper identities and version control. Testing continuously will in practice require automation which means that we need accurate and valid test data for each kind of test and we need to be able to handle the test data automatically regardless of whether we have to generate it every time or if we can store and reuse it. We also need to be able to later identify what test data was used for each test when we want to rerun the same test for root cause analysis, regression testing etc.

Monitor and feedback loop activities will generate data and information that we later want to visualise, aggregate, use for prioritisation, use for innovation etc. By handling this type of information as configuration items, we get the ability to store it properly and connect it both to our product and to activities in our development work.

The DevOps activities have a high degree of automation to be efficient and valuable. Some of the artefacts created are of temporary value while others live with the product and need to be defined as configuration items. This means that we need automated support also for configuration identification. It is important that we can create new identities, new versions, store new meta data and register new dependencies for our configuration items and that our Configuration Management Database is prepared to handle all the types of configuration items that we can foresee. It must also be easy to instantiate new types of configuration items when our way of working evolves.

Software development today involves a high degree of inclusion of third-party products and libraries, sometimes from the open-source community, sometimes commercial parts. It is important to realise that every item that we include as a part of our product and/or in our development work will become a configuration item in our system. This can be implemented either as an external reference with identity and version, or by storing them in our own repositories. Handling those artefacts as configuration items will also make it possible for us to include them in configurations and baselines just as any other part of the product.

4.1.3 Configuration control & Change management – *supports: planning, change management devops, monitor and sharing*

Planning might not be the first activity that comes to mind when you hear “DevOps”, but to be able to establish a DevOps culture along with good tool support you need planning for sure. Involving the SCM expert in the planning will ensure that the DevOps related activities maintain traceability, reproducibility and full control of what changes are included vs excluded in tests and deliveries.

When we in a DevOps context talk about change management, we focus on how to determine what changes should be made and not. Once a code change is done, all decisions and considerations should already have been made. In a traditional way of working, change management commonly includes a Change Control Board meeting decision to authorize that the change can be released. To align with the DevOps culture, we need to change the procedure and make the Change Control Board be proactive and make the release decision already while the change is being implemented. When using a pipeline to build, integrate and

test changes it becomes obvious that the upfront release decisions must be made with the reservation that the change passes the pipeline without problems. DevOps does not mean that we start compromising on product quality. We can also “protect” release branches and/or release tags to make sure that changes that don’t pass the release criteria cannot be included in a released product and reduce the risk of human errors.

Many DevOps practising organisations use feature flags for controlling what features should be available in the product. Feature flags give us the opportunity to control in runtime what features are active and can both simplify and complicate the CM work at the same time. Using feature flags makes it possible to work with a simplified branching strategy (we might go all the way to trunk-based development), but it will also add the complexity of keeping track of the feature configuration in the runtime environment for the product. The feature configuration will become configuration item(s) in the runtime environment and must be possible to identify in baselines and change management during operations. It is important to realise the importance of traceability and control of the runtime configuration(s) and include this information in the feedback loops to provide the development organisation with the data they need for problem analysis.

When monitoring the product in its production environment it is important to know what changes are deployed and active. We may for example want to monitor a new feature separately and measure how and how much it is being used. The same goes for when we are sharing information about the product. The receiver of the shared information must be able to determine what features are implemented and what errors have been corrected.

4.2 DevOps characteristics being supported by SCM activities

Now we switch perspective and start from some of the DevOps characteristics point of view to see how they can get support from the configuration manager. This is food for thoughts for a DevOps practitioner that wonders what help to ask for from the SCM expert. For the configuration manager, this section gives examples of DevOps characteristics to look for in the organisation and investigate if the SCM related problems there are solved.

4.2.1 CI/CD/Continuous Deployment – *supported by: Configuration Audit, Build Management, Release Management*
“Continuous” means that we need automation. We need to automate the pipeline for building, testing, delivering and deploying to as high a degree as possible. Once it is easier to build, test and deploy changes, developers will use these capabilities to ensure the quality of their changes. We will therefore need to be able to handle an increased number of product versions, variants and configurations.

When automating the pipeline for deployments or deliveries, we need to move all manual activities to take place either before the code is done or after the delivery/deployment is made. CM intense activities like audits, release decisions etc need to be reshaped to a form where they don’t act as impediments in the pipeline or in the feedback loops. Release decisions can for example be made with reservation for quality and security criteria that can be automatically verified in the pipeline.

4.2.2 Monitoring – *supported by: Configuration Identification, Change management, Configuration Status Accounting, Version Control*
To be able to monitor the product in its production environment, gather usage data and use it to determine and prioritise future work we will need to be able to identify what product version and configuration the data comes from. This kind of traceability requires both proper configuration identification and version control in place as well as change management to know what features and bug fixes are implemented in the configuration from where the gathered data origins. It is here not only the product that needs a formal identity but also the collected data to be able to store it and connect it to the product configuration.

Presenting the data in an accessible way is essential to turn it into information. This is basically a natural part of modern configuration status accounting and it is often wise to gather all data visualisation in the same platform. In a DevOps context, it should also be a natural step to automate the monitoring to achieve a continuous information flow in the feedback loop(s).

4.2.3 Change management (DevOps) – *supported by: Configuration Identification, Configuration Control, Change Management*
When continuous delivery or continuous deployment is established, there isn't much room for manual change management in the output flow. Decisions about changes need to be made up front with criteria for when those changes can be included. The idea is that by establishing proper quality criteria and definition of done that applies to all changes, we will be able to determine in advance when a change should be integrated to what release. Once the code is done and the quality criteria are fulfilled (controlled by the automated pipeline), the change can then be delivered without any further delays.

Using feature flags for controlling what changes should be active in the production environment of the system is used more and more often in DevOps practicing organisations. Also, this way of managing changes will need traceability and control and can be supported by SCM practices as described in section 4.1.3.

Up front decisions and planning like this requires proper configuration identification. criteria for distributed, or even automated, decisions require good support in the tool chain for configuration control and change management and we will of course need good traceability to be able to track which changes were included when. Automated support for creating new configuration items, and versions is essential but also support for creating new baselines automatically and managing the status for issues and other backlog items.

As described above in section 4.1.3, the change management procedures will be highly affected by this way of working with changes and the CCB will need to act in a more proactive way with changes than in a traditional setup.

4.3 Summary

It is quite obvious that there are several practical areas where the SCM expert and the DevOps practitioner can add value and make life better for the developers by collaborating and make sure that they are aligned in how to create a work environment that enables both continuous deliveries, continuous deployments, fast feedback loops, continuous monitoring and supports continuous control of the product quality.

The role of the SCM expert in a DevOps context will be different than in a traditional more waterfall-oriented organisation. The role will be more of a true expert role and the operational SCM tasks will be performed either automated in pipelines and tools or by the development teams. Handovers between requirements writers, developers, testers and operations will be carried out within the same team. The job for the SCM expert will be to make sure that the teams have the right knowledge, tool support and authorities to develop and deploy the software themselves while keeping the desired level of traceability, change control and maintainability.

5. Conclusions

In this white paper, we have described a model that maps the detailed characteristics of DevOps to the corresponding SCM activities – and vice-versa. We have also given some general guidelines and concrete examples for how relevant SCM concepts and principles can be operationally implemented in a DevOps context.

Given that our initiating problem was “what SCM activities are needed in a DevOps context and how to carry them out”, we can use these two results to give a solution to the initiating problem. As demonstrated from the model in figure 3.1 and detailed in [HS, 2020] the first result clearly shows that all SCM activities are able to support one or more DevOps activities. So, all parts of SCM will be useful also in a DevOps context. The model also shows that all identified DevOps characteristics can get some kind of support from one or more SCM activities.

However, some things will be different in a DevOps context. The approach to the activity “SCM plan” will change. Instead of a fixed plan up front, it should be a continuous effort where it is created incrementally, “as needed” and as “live documents”. The type of SCM work will also change. Since we have cross-functional DevOps teams/developers they will take care of the operational SCM work. This will leave the SCM expert with strategic SCM work like giving the DevOps practitioners the tools and the knowledge they need. We also showed how SCM practices can be adapted and automated to better fit a DevOps oriented organisation. It shows that it is quite simple and straight-forward to adapt and implement the more abstract concepts and principles of SCM in a DevOps context.

We have demonstrated that Software Configuration Management concepts and principles are certainly relevant and useful also in a DevOps context and that they can be made operational.

6. References

[Abildskov, 2020]: Johan Abildskov: *What is DevOps?*, blog-post, EfiCode, July 2020, https://www.eficode.com/blog/what-is-devops (retrieved August 19, 2021).

[Bass et al., 2015]: Lenn Bass, Ingo Weber, Liming Zhu: *DevOps: A Software Architect's Perspective*, Addison-Wesley Professional, 2015.

[BA, 2005]: Kent Beck, Cynthia Anders: *Extreme Programming Explained: Embrace Change*, 2nd edition, Addison-Wesley Professional, 2005.

[Forsgren et al., 2018]: Nicole Forsgren, Jez Humble, Gene Kim: *Accelerate – Building and Scaling High Performing Technology Organizations*, First edition, IT Revolution, 2018.

[HS, 2020]: Erik Hochbergs, Laroy Nilsson Sjödahl: *Software Configuration Management in a DevOps context*, Master's thesis, Department of Computer Science, Lund University, Sweden, January 2020.

[Mann et al., 2018]: Andi Mann, Michael Stahnke, Alanna Brown, Nigel Kersten: *State of DevOps Report – 2018*, presented by puppet + splunk, https://puppet.com/resources/report/2018-state-devops-report (retrieved August 19, 2021).

[Wikipedia, 2021]: Wikipedia: *DevOps*, https://en.wikipedia.org/wiki/DevOps (retrieved August 19, 2021).

Acknowledgements
We would like to thank our reviewers (Johan Abildskov, Fred Jonkhart, Mauro de Pascale, Kasper Østerbye and two who wanted to remain anonymous) for their great work. Their remarks and suggestions really made us think and reflect on what it was we wanted to communicate and how to communicate it. We did not accept all change requests since they in some cases – even if very valid – would give this white paper a different or broader focus than what we wanted. So, any remaining flaws are entirely the responsibility of the authors.

During the first part of the work on this white paper Christian Pendleton was with Eficode AB, Malmö, Sweden.

Appendix
Corresponding author:
Lars Bendix: bendix@sneSCM.org

Given that this is version 1.0 there might be a version 1.1 – or there might be another white paper with a different scope. Your feedback and input is highly valued, so if you find issues with this white paper, please send us a change request.

"Change request form":
Where (paper, chapter, section, paragraph, line)?
What (description of "problem")?
Comments (possible suggestion for fix, if you have some – otherwise leave it to us)?
Severity (Must fix, Should fix, Could fix, Language issue)?