Dimensions of Software Configuration

On the Configuration Context in Modern Software Development

ESEC/FSE 2020
8–13 November 2020

Norbert Siegmund
@Norbsen

Leipzig University

Nicolai Ruckel
@NicolaiRuckel

Bauhaus-Universität Weimar

Janet Siegmund
@JanetSiegmund

Chemnitz University of Technology

Hi, my name is Nicolai Ruckel and welcome to my presentation about *Dimensions of Software Configuration: On the Configuration Context in Modern Software Development*. This is a joint work with Norbert Siegmund from Leipzig University and Janet Siegmund from Chemnitz University of Technology.

“A simple example [of configuration] is when I start a Spring Boot application and this works standalone, then it is totally trivial. As soon as I start booting multiple of them, the traditional way would be to take Spring Cloud. [...] But then every Spring Boot application taken into Spring Cloud has a properties file with not 2 but 50 entries. All configuration.” — I$_1$

I would like to start this talk with a quote from a software consultant we talked to: “A simple example [of configuration] is when I start a Spring Boot application and this works standalone, then it is totally trivial As soon as I start booting multiple of them, the traditional way would be to take Spring Cloud. [...] But then every Spring Boot application taken into Spring Cloud has a properties file with not 2 but 50 entries. All configuration.”

Configuration is Everywhere

Configuration is Everywhere

Configuration is Everywhere

Configuration is Everywhere

IJ

eclipse

Maven™

spring®

Configuration is Everywhere

IJ

eclipse

Maven™

NGINX

spring®

Configuration is Everywhere

IJ

eclipse

mongoDB

Maven™

NGINX

redis

spring

Configuration is Everywhere

IJ

eclipse

Maven™

mongoDB®

NGINX

redis

spring®

Configuration is Everywhere

2

Like he said, configuration is an important topic in both research and industry. For instance, we configure the IDE in which we are writing the code, we configure libraries and frameworks, we configure build files with diverse tools, we configure the environment, in which our application is executed, we configure database connections, ports, and tests of our application. Recently, we configure the whole CI/CD pipelines. This also involves the configuration of the infrastructure—for instance via Kubernetes—on which our software system is possibly distributed.

Configuration Can Mean Different Things

Configuration Can Mean Different Things

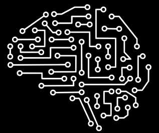
Test	P1	P2	P3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	1	1	0

Input variables and parameters

Configuration Can Mean Different Things

Test	P1	P2	P3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	1	1	0

Input variables and parameters



Specification of experiment

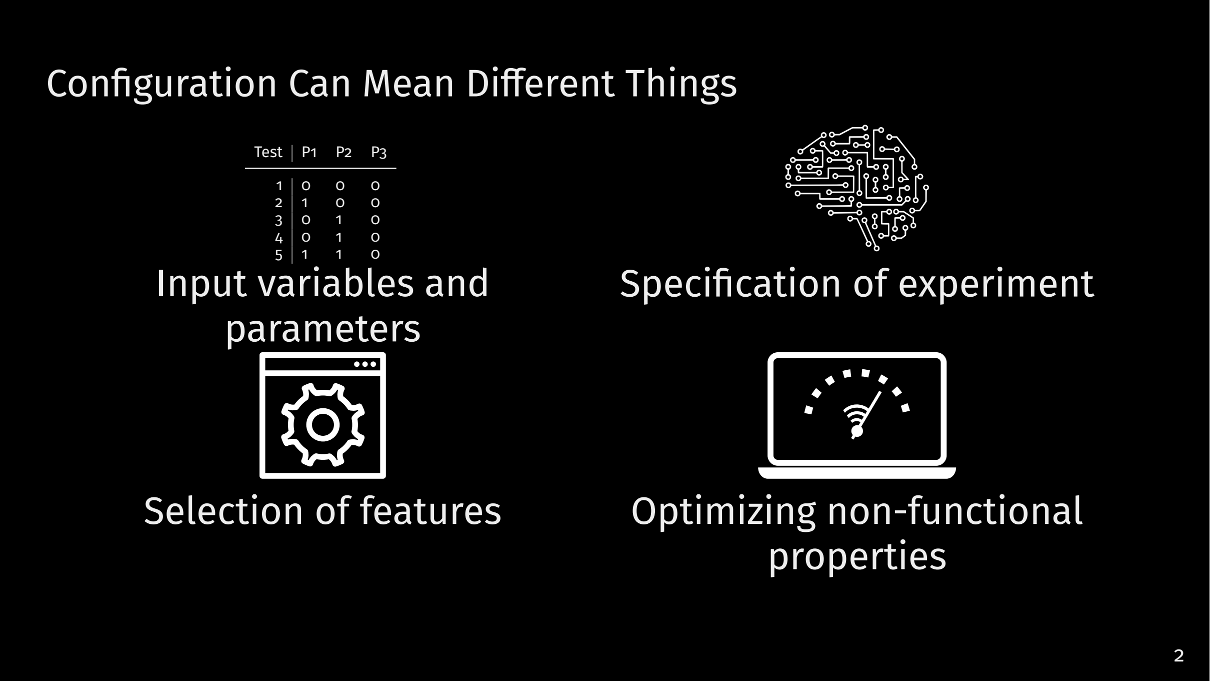
Configuration Can Mean Different Things

Test	P1	P2	P3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	1	1	0

Input variables and parameters

Specification of experiment

Selection of features



Configuration Can Mean Different Things

Test	P1	P2	P3
1	0	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	1	1	0

Input variables and parameters

Specification of experiment

Selection of features

Optimizing non-functional properties

Despite its importance, there is no clear definition of the term configuration and therefore each research paper assumes a different context in which configuration is involved. Often, this context depends on the domain and is just implied and not directly specified. For example, in combinatorial testing configuration refers to input variables and parameters and in a machine-learning setting this would specify an experiment. In software product lines, configuration refers to a selection of features customizing the functional aspects of the software variants. In other domains such as performance optimization, configuration is instead focused on finding software configuration that improves non-functional properties.

Consequences

- Results are
  - difficult to compare and generalize

Consequences

- Results are
  - difficult to compare and generalize
  - not directly transferable to practice

Consequences

- Results are
  - difficult to compare and generalize
  - not directly transferable to practice
- Research is often missing context

This means that it is difficult to compare or generalize the results from papers from different domains involving configuration. Also, insights and results are often not directly transferable to practice. In research the context is often not specified at all. Without that context it is unclear how to use the results.

Example: Combinatorial Testing

Example: Combinatorial Testing

- Aims at finding bugs in a program

Example: Combinatorial Testing

- Aims at finding bugs in a program
- Uses different sets of inputs and options

Example: Combinatorial Testing

- Aims at finding bugs in a program
- Uses different sets of inputs and options

Problems:

- Not all options are available at all time

Example: Combinatorial Testing

- Aims at finding bugs in a program
- Uses different sets of inputs and options

Problems:

- Not all options are available at all time
- Availability of data depends on deployment stage

Example: Combinatorial Testing

- Aims at finding bugs in a program
- Uses different sets of inputs and options

Problems:

- Not all options are available at all time
- Availability of data depends on deployment stage
- Knowledge about configurations is distributed among stakeholders

Let us consider the example of combinatorial testing. Combinatorial testing aims at finding bugs in a program by covering different combinations of input parameters and options. However, not all options and settings are available at the same time due to step-wise configuration process. Also, the availability of data for testing depends on the deployment stage whereas knowledge about configuration settings is distributed over a diverse set of stakeholders.

Goals

Type

Goals

Type

Binding Time

Goals

Type

Binding Time

Stakeholder

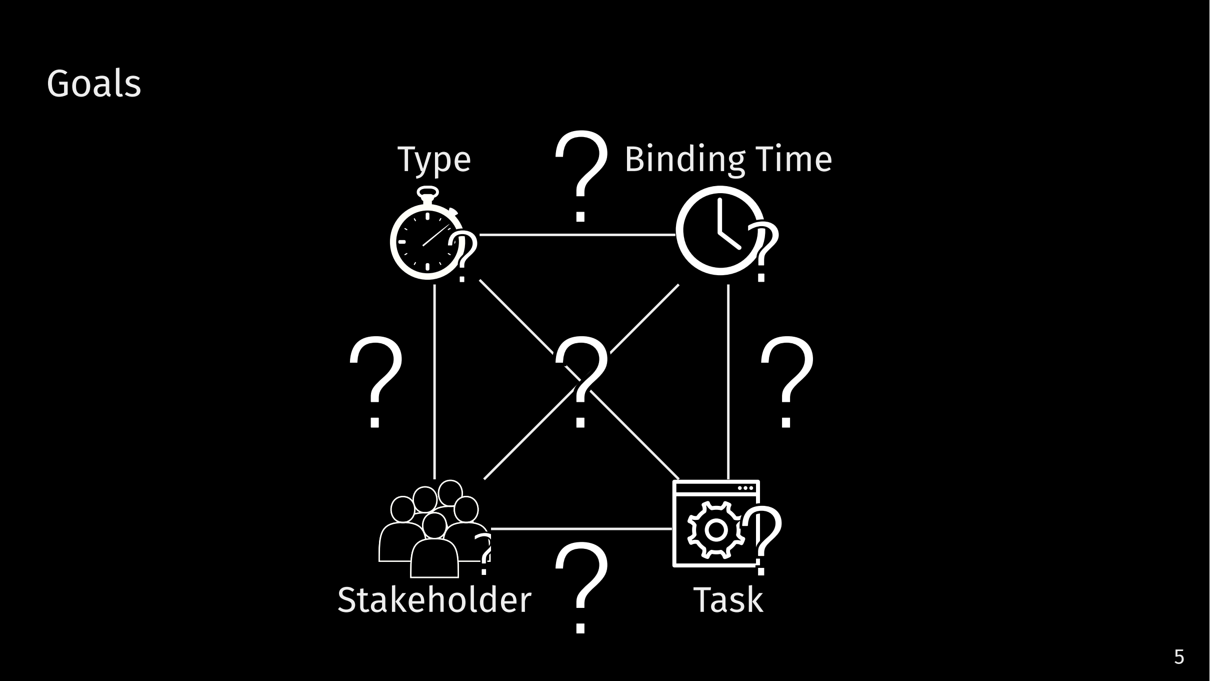
Goals

Type

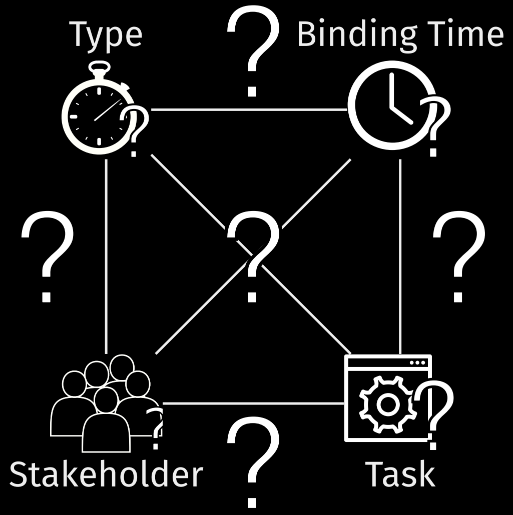
Binding Time

Stakeholder

Task



Goals



In essence, we do not know which types of configuration exist, which binding times of configuration values are prevalent for which types of configurations, which stakeholders are involved in configuration, and what tasks during software development require configuration. Furthermore, all these factors interact, but we do not know how. Thus, we set out to shed light on which factors comprise configuration and how they interact.

Goals

Goals

- Find all aspects of configuration

Goals

- Find all aspects of configuration
- Provide framework for configuration to

Goals

- Find all aspects of configuration
- Provide framework for configuration to
    - understand interacting aspects of configuration

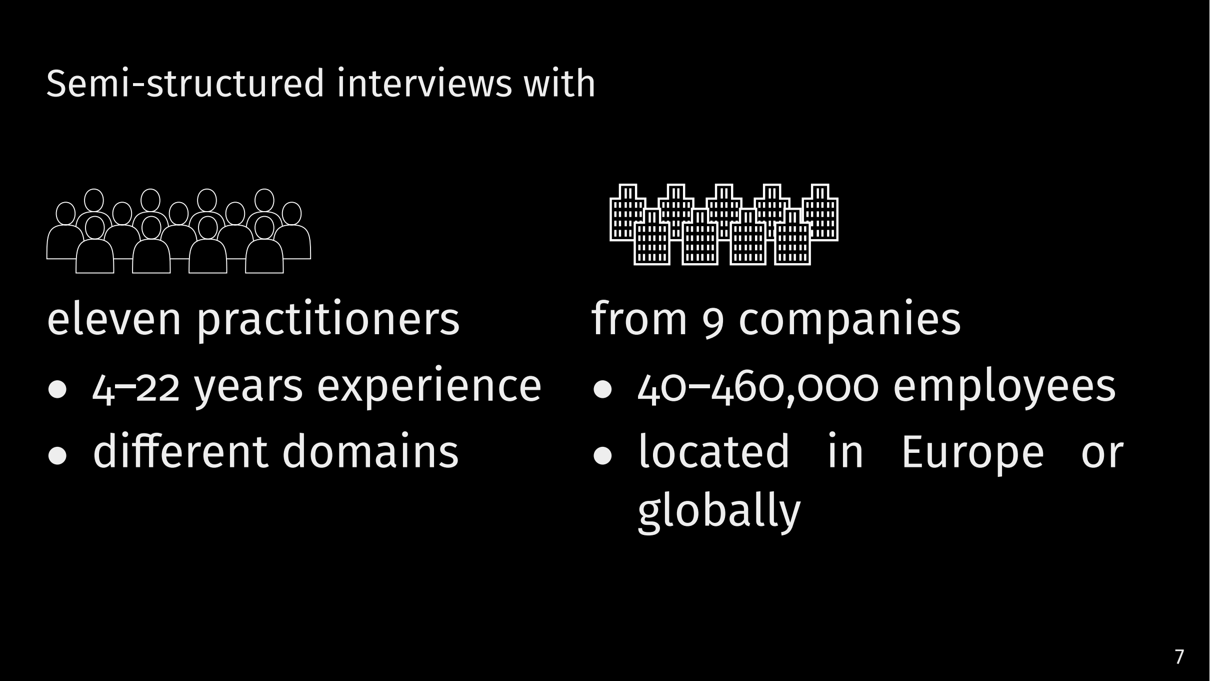
Goals

- Find all aspects of configuration
- Provide framework for configuration to
  - understand interacting aspects of configuration
  - guide future research

Goals

- Find all aspects of configuration
- Provide framework for configuration to
  - understand interacting aspects of configuration
  - guide future research
  - derive implications in practice

Our goal is to frame the term configuration, find out all aspects involved in configuration to have a common terminology. This terminology helps with building a comprehensive model on the aspects of configuration. We want to provide a framework for researchers to place their work and studies into context and derive what factors need to be considered.
This helps understand how research results apply to which interacting aspects of configuration, including their limitations and relevant circumstances in practice. Furthermore, it can guide research of each isolated area to ask the right questions, and also how to combine these areas to increase practical relevance. For practitioners, the framework can be a means for orientation, such that stakeholders can place their current configuration activity in the model and derive possible interactions with other stakeholders and implications for them.



Semi-structured interviews with

eleven practitioners

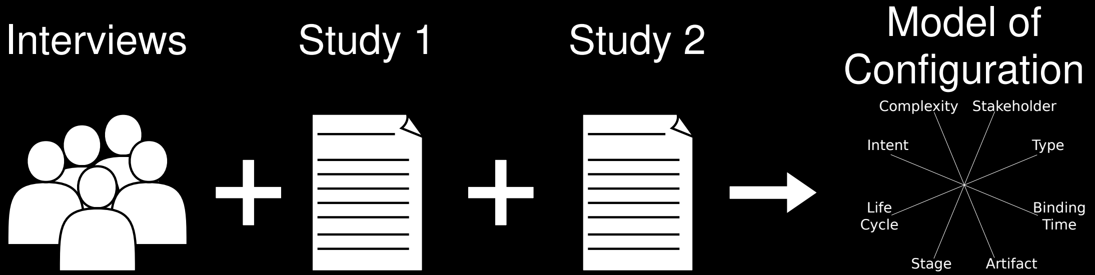
- 4–22 years experience
- different domains

from 9 companies

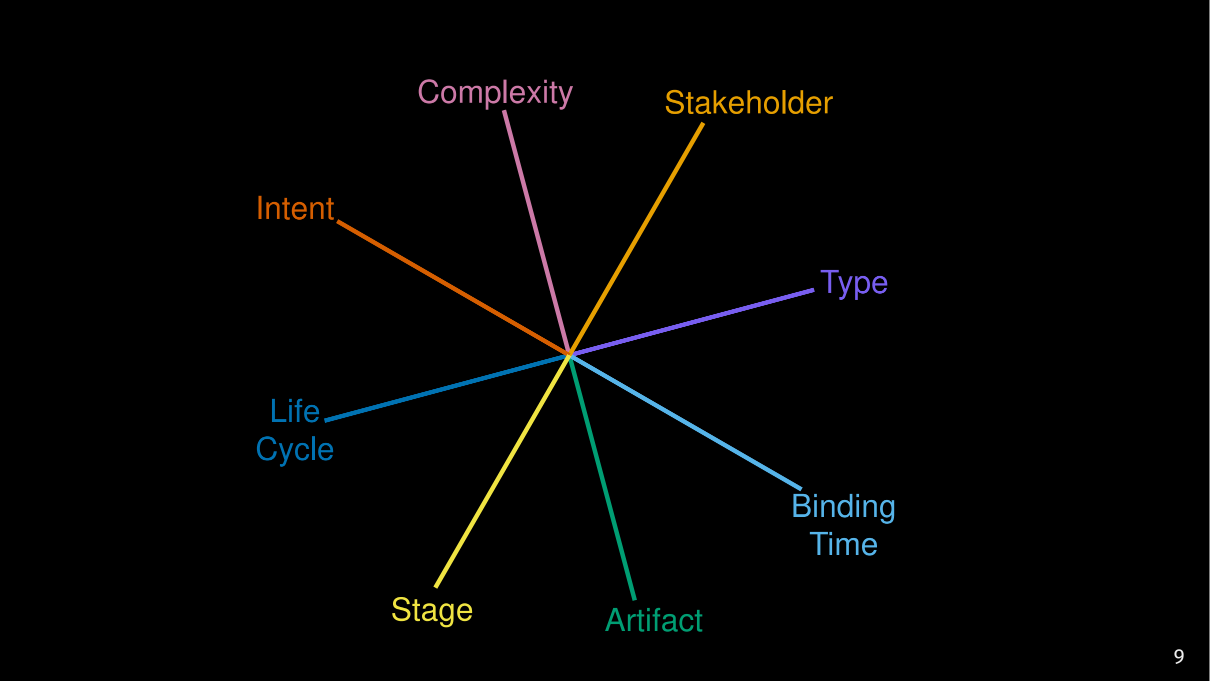
- 40–460,000 employees
- located in Europe or globally

To achieve that, we conducted semi-structured interviews with 11 practitioners from nine companies coming from different domains such as DevOps, microservices, or frontend with different levels of experience, ranging from 5 to 22 years.
We followed a grounded theory approach for the study. That means, instead of focussing on specific questions we asked general questions about configuration such as “What is configuration for you?” in the beginning and became more specific in later questions and interviews. We transcribed the interviews and analyzed the answers with a card-sorting approach.

Methodology

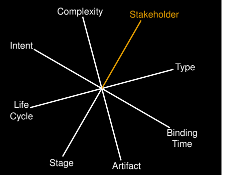


Afterwards, we looked into two other studies about configuration to compare our results with theirs. With those insights, we built a model of configuration.

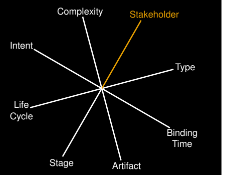


This model of configuration is the main result of our study. It comprises different dimensions of configuration. After considering the additional literature we found eight different dimensions of configuration. These dimensions are stakeholder, type, binding time, artifact, stage, life cycle, intent, and complexity. Next, I will explain each dimension and their values.

Stakeholder

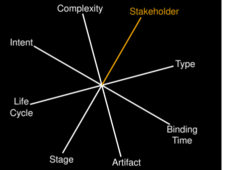


Stakeholder



Values: Developer, Operator, DevOps, Technical User, End User

Stakeholder

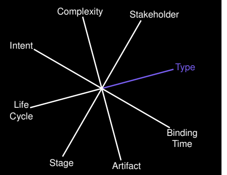


“With the DevOps approach, requirements have changed [...]. [...] they not only need to understand the technology [...], but also give recommendations for configurations or apply configurations on their own and take responsibility for them.” — I_8

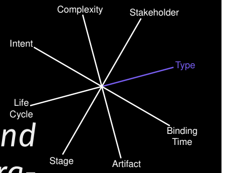
Values: Developer, Operator, DevOps, Technical User, End User

Let us first look at the stakeholder dimension. The involvement of different stakeholders from different backgrounds in the configuration process strongly influences how configuration is presented, validated, and maintained. We found stakeholders to be developers, operators, DevOps, technical users, and end users. Of course all of those stakeholder categories can be further divided to show role specific aspects, for example frontend or backend developers. For example operators configure underlying hardware systems, operating systems or other infrastructure, while developers create the configuration options and provide default values. With the rise of DevOps the separation between those two groups has been blurred. One of the interviewees said “With the DevOps approach, requirements have changed [...]. [...] they not only need to understand the technology [...], but also give recommendations for configurations or apply configurations on their own and take responsibility for them.”

Configuration Type

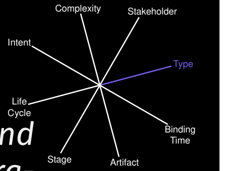


Configuration Type



“The domain-specific configuration helps us […] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong […].” — I$_7$

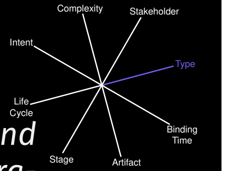
Configuration Type



"The domain-specific configuration helps us [...] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong [...]." — I_7

Infrastructure

Configuration Type

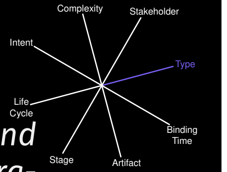


“The domain-specific configuration helps us […] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong […].” — I_7

Infrastructure

Development

Configuration Type

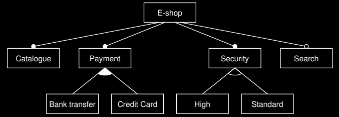


“The domain-specific configuration helps us [...] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong [...].” — I_7

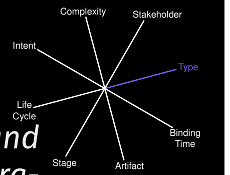
Infrastructure

Development

Domain



Configuration Type

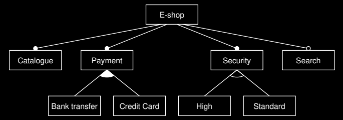


“The domain-specific configuration helps us […] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong […].” — I_7

Infrastructure

Development

Domain

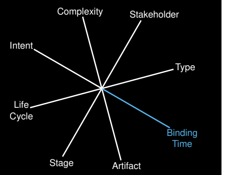


Values: Development, Infrastructure, Domain

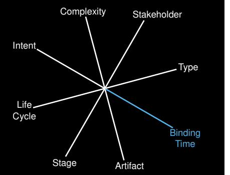
The interviewees identified two main types of configuration: domain-specific and technical configuration which we modeled in this dimension. One practitioner described it as follows. “The domain-specific configuration helps us […] to faster provide and customize functions to different customers. And technical configuration can break many things if done wrong and would need more attention.”

Technical configuration comprises infrastructure configuration and development configuration, as the kind of tools, configuration artifacts, and configuration effort differ substantially. Infrastructure configuration refers to adjusting a software system to the underlying hardware and software, such as connecting to the correct database system, using a specific port, or setting environmental variables in a Docker file. Development configuration as stated in the interviews involves setting up development tools, such as IDEs, and build tools, the build and testing process, and automating the deployment process to different stages.

Binding Time

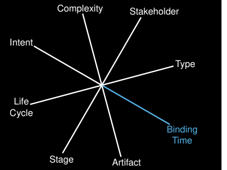


Binding Time



“Usually, microservices are relatively fast to deploy, so that, most of the time, dynamic configuration is not needed. So, you reconfigure something in a file and redeploy the container.” — I_7

Binding Time

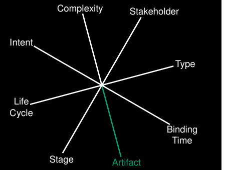


"Usually, microservices are relatively fast to deploy, so that, most of the time, dynamic configuration is not needed. So, you reconfigure something in a file and redeploy the container." — I_7

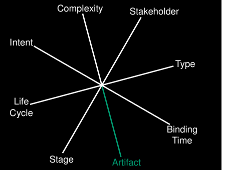
Values: Build, Deployment, Load Time, Run Time

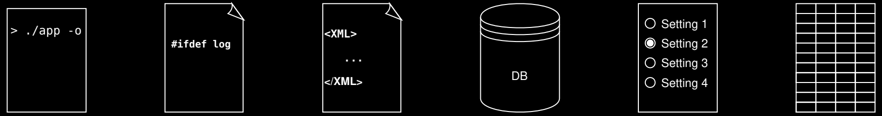
Binding time refers to the event of binding a configuration option to a certain value. The binding time of an option varies largely, from depending on build time for example in form of build scripts, deployment time in form of configuration files, to load- and run time. Hence, binding time strongly interacts with the dimension's stakeholder, type of configuration, and configuration artifact. It also depends on the domain as one of the interviewees stated: "Usually, microservices are relatively fast to deploy, so that, most of the time, dynamic configuration is not needed. So, you reconfigure something in a file and redeploy the container." Thinking about an appropriate binding time of configuration options is an undervalued task in academia and practice as most research papers don't specify this.

Configuration Artifact



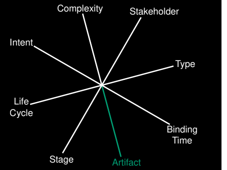
Configuration Artifact

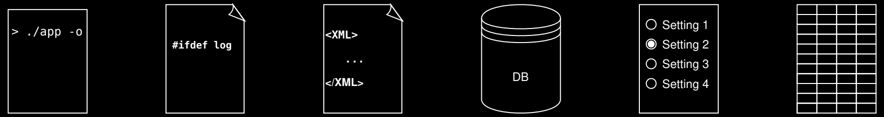




Values: Source Code, Configuration File, Database, Command Line Parameter, ...

Configuration Artifact



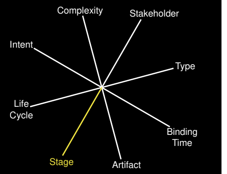


"Configuration should be easy. It should have an easy format that is also comprehensible and that is easily readable by humans and has possibly few indirections." — I_7

Values: Source Code, Configuration File, Database, Command Line Parameter, ...

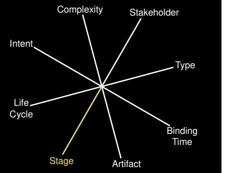
Nowadays, configurations are distributed in many artifacts that exhibit their own structure, syntax, and semantic. This is described by the configuration artifact dimension. Even though different formats have different advantages and disadvantages, we found that simple configuration files, such as properties or ini files are favored by our interviewees. They said that "[configuration] should have an easy format that is also comprehensible and that is easily readable by humans and has possibly few indirections".

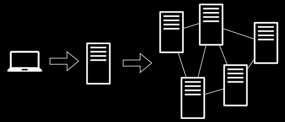
Stage



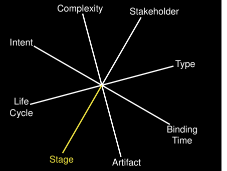
Complexity
Stakeholder
Intent
Type
Life Cycle
Binding Time
Stage
Artifact

Stage

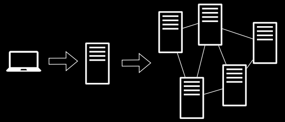

Complexity
Stakeholder
Intent
Type
Life Cycle
Binding Time
Stage
Artifact



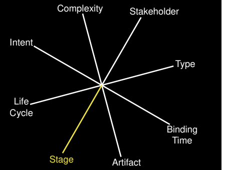
Stage



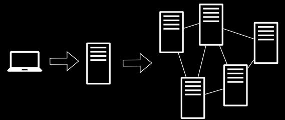
“So, you have your application, which you can configure. Then you have your application in the environment, in which it is deployed, and there is an additional configuration that diverges.” — I_3



Stage



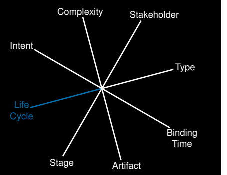
"So, you have your application, which you can configure. Then you have your application in the environment, in which it is deployed, and there is an additional configuration that diverges." — I_3



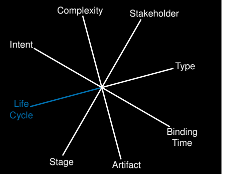
Values: Development, Test, Pre-production, Production

The stage dimension describes that configuration happens in different stages of the development process, such as development or testing. Each stage usually describes a different infrastructure environment of the running system, variables, such as the JVM class path, and available resources, such as database systems. Additionally, it is often strongly connected to a stage in the CI/CD process, in which a software system is deployed and executed. These stages lead to different configurations as one of the practitioners put it when he said: "So, you have your application, which you can configure. Then you have your application in the environment, in which it is deployed, and there is an additional configuration that diverges." Typical stages include testing, pre-production, and production.

Life Cycle

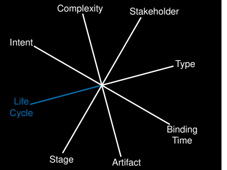

Complexity
Stakeholder
Intent
Type
Life Cycle
Binding Time
Stage
Artifact

Life Cycle



Values: Create, Maintain, Bind, Own, Deprecate, Remove

Life Cycle

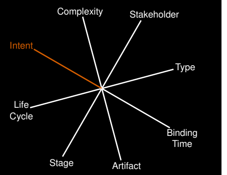


“[...] I can customize my application server with a certain VM size. Which VM size is later actually applied, we don’t know. [...] we have tested it with this minimum and that maximum. But the final configuration is done by the Ops guys [...]. And thus they own the values and are responsible for them.” — I_8

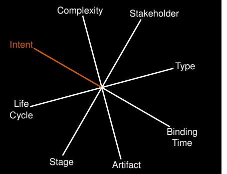
Values: Create, Maintain, Bind, Own, Deprecate, Remove

This dimension of life cycle describes the diverse aspects of configuration options from creation and maintenance over binding to deprecation, that is, all lifetime phases. One of our interviewees told us “[...] I can customize my application server with a certain VM size. Which VM size is later actually applied, we don’t know. [...] we have tested it with this minimum and that maximum. But the final configuration is done by the Ops guys [...]. And thus they own the values and are responsible for them.” That is, every stakeholder is responsible for different lifetime phases and affected by the earlier ones. Although some research has identified diverse problems when introducing more and more options, it is often not clear that a configuration option has its own life cycle, which starts in the requirements phase when, for example, a new optional feature is planned.

Intent

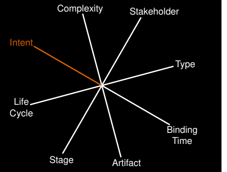

Complexity
Stakeholder
Intent
Type
Life Cycle
Binding Time
Stage
Artifact

Intent



Values: A/B Testing, Code Reuse, Knowledge Preservation, Distributed Environment, ...

Intent

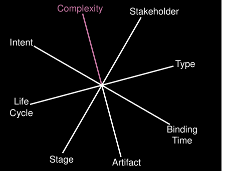


“Due to the team’s autonomy, it is desired that when you have implemented a feature, you can take it to production, but it is deactivated with a toggle. But it is already present in production code. If all teams have finished their [dependent] implementations, then each needs only to turn on the switch to activate the feature.” — I_9

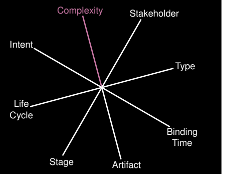
Values: A/B Testing, Code Reuse, Knowledge Preservation, Distributed Environment, ...

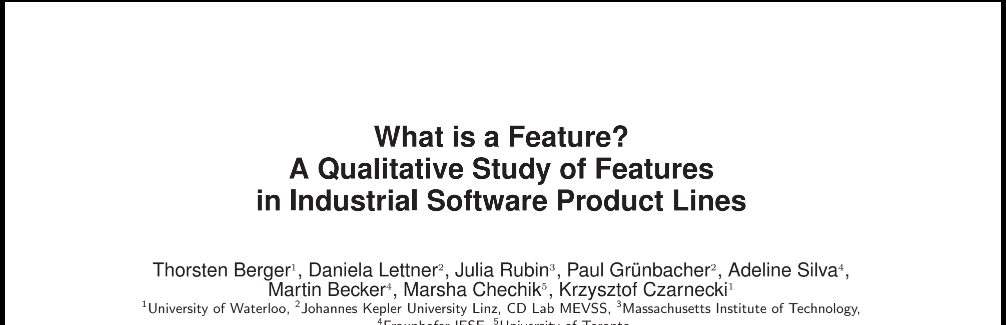
This dimension describes why configuration options are introduced in the first place, which can have a strong effect on the other dimensions. We found that often, options are introduced to tailor functional behavior, to tailor non-functional behavior, and to reuse code. However, some interviewees pointed out that there exist several more intents that trigger the implementation of configuration options such as knowledge preservation or to be able to make a software system runnable on an unknown environment. Another example is to coordinate the deployment of a new feature that spans over multiple services like in the case of one practitioner who said “Due to the team’s autonomy, it is desired that when you have implemented a feature, you can take it to production, but it is deactivated with a toggle. But it is already present in production code. If all teams have finished their [dependent] implementations, then each needs only to turn on the switch to activate the feature.”

Complexity


Complexity
Stakeholder
Intent
Type
Life Cycle
Binding Time
Stage
Artifact

Complexity

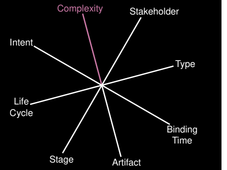




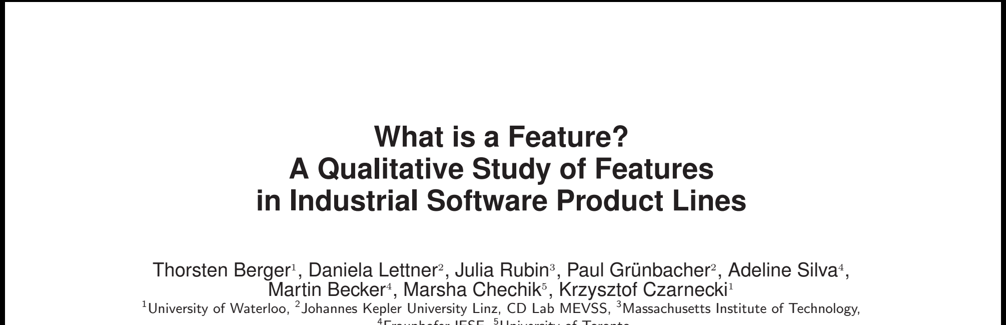
What is a Feature?
A Qualitative Study of Features
in Industrial Software Product Lines

Thorsten Berger[1], Daniela Lettner[2], Julia Rubin[3], Paul Grünbacher[2], Adeline Silva[4], Martin Becker[4], Marsha Chechik[5], Krzysztof Czarnecki[1]
[1]University of Waterloo, [2]Johannes Kepler University Linz, CD Lab MEVSS, [3]Massachusetts Institute of Technology,
[4]Fraunhofer IESE, [5]University of Toronto

Complexity



Values: Local Scope, Distributed Scope, High Dependency, Low Dependency, No Dependency


What is a Feature?
A Qualitative Study of Features
in Industrial Software Product Lines

Thorsten Berger[1], Daniela Lettner[2], Julia Rubin[3], Paul Grünbacher[2], Adeline Silva[4], Martin Becker[4], Marsha Chechik[5], Krzysztof Czarnecki[1]
[1]University of Waterloo, [2]Johannes Kepler University Linz, CD Lab MEVSS, [3]Massachusetts Institute of Technology,

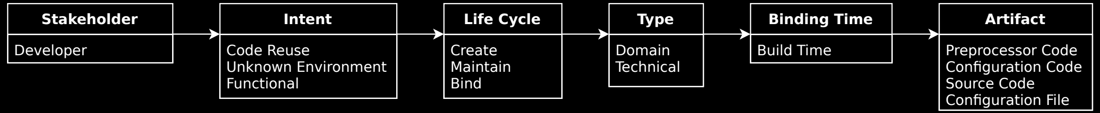
Although there are many aspects contributing to the configuration's complexity, there are certain values that need special attention. So far, we have not explicitly addressed the scope of a configuration with respect to software modules and artifacts. The interviewees of Berger et al. reported on configuration that affects only individual modules and configuration that is distributed among different modules, frameworks, services, and other artifacts. Moreover, an option or entire configuration might be dependent on other options and configurations at different degrees. For example, the presence and valid values of an option might depend on the configuration of its cloud infrastructure or which customer owns the product. In such cases, the complexity increases and interacts with other dimensions, such as the configuration artifacts, stage, and type. So, choosing the right artifact for a specific type of configuration depends on the complexity of the configuration.

Application of the Model

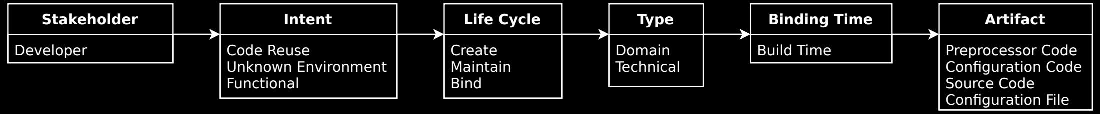
To show our model in action, we provide a model of configuration for 18 research papers selected from three related research fields: configuration engineering, software product lines, performance optimization, testing, and configuration errors. Here, I want to talk about the insights in software product lines and performance optimization.

Software Product Lines

Software Product Lines

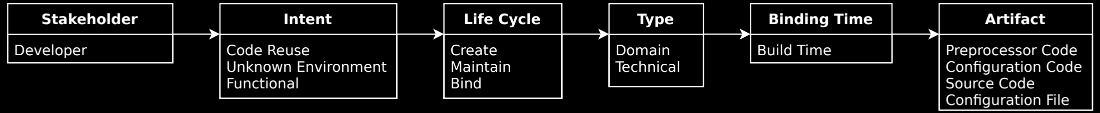


Software Product Lines



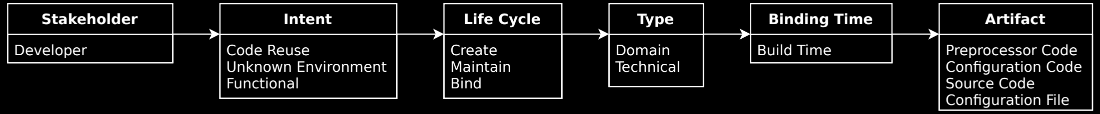
- No DevOps

Software Product Lines



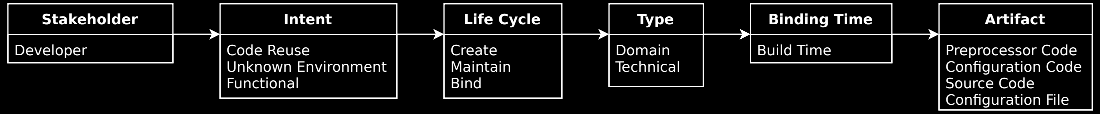
- No DevOps
- No removing or deprecation of options

Software Product Lines



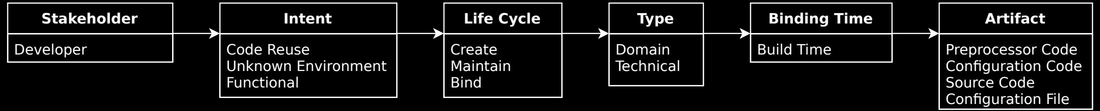
- No DevOps
- No removing or deprecation of options
- Effect of different binding times on implementation techniques

Software Product Lines



- No DevOps
- No removing or deprecation of options
- Effect of different binding times on implementation techniques
- No modern staged deployments and cloud infrastructures

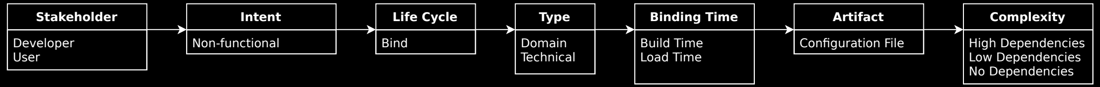
Software Product Lines



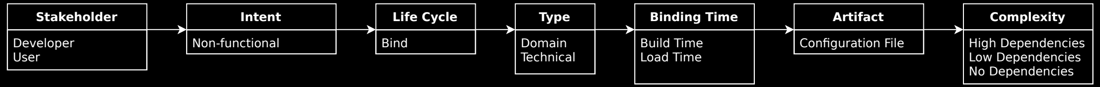
- No DevOps
- No removing or deprecation of options
- Effect of different binding times on implementation techniques
- No modern staged deployments and cloud infrastructures
- No complexity analysis with respect to SPL implementation techniques

Let's start with software product lines. We analyzed several papers in the area of software product line research to see which dimensions and values are covered. The model shows that those papers mainly focused on developers, which is natural as mostly implementation techniques are discussed. However, it appears that the new trend of DevOps seems to not have reached yet research in this area. Also not surprisingly is that most papers focus around code reuse and functional customization of software systems. What is missing in this context is (i) evolution of product lines, especially how to deprecate and remove configuration options (life cycle), (ii) different binding times and their effect on implementation techniques, (iii) modern staged deployments and cloud infrastructures for product line development (stage and type), and (iv) complexity analysis with respect to product line implementation techniques.

Performance Optimization

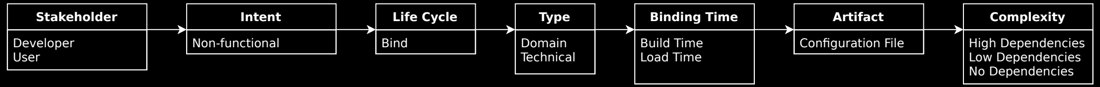


Performance Optimization



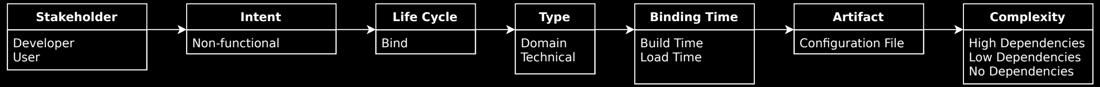
- Unknown Stage

Performance Optimization



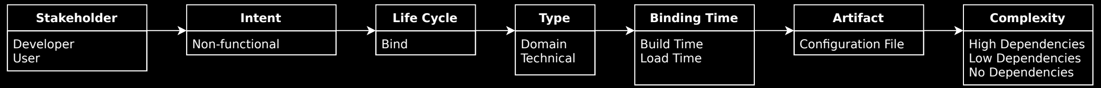
- Unknown Stage
- Infrastructure ignored

Performance Optimization



- Unknown Stage
- Infrastructure ignored
- Reconfiguration and evolution not considered

Performance Optimization



- Unknown Stage
- Infrastructure ignored
- Reconfiguration and evolution not considered
- No DevOps or Ops

Next, we look at performance optimization. The papers we chose here all take the same context: A user or developer configures a software system via a configuration file either at load or build time. None of the papers made the environment explicit. It is unknown at which stage the performance was optimized and the infrastructure is not taken into account even though it can have a profound influence on performance. With the restriction on binding configuration options, reconfiguration or software evolution is not considered. Moreover, performance is mainly driven by the underlying hardware. The persons that are responsible for configuring this hardware—namely DevOps and Ops people—should also be considered to make the results more practical.

Best Practices

In addition to our model we found best practices for dealing with configuration in the industry.

Avoiding Configuration Errors

Avoiding Configuration Errors

“Validation is part of the meta-configuration. [...] So, there is a kind of management model, that is, a configuration option is not only the name of the option, but also the data type, a validation, or where does this configuration value come from, so cross relations to other configuration values.” — I_{11}

Avoiding Configuration Errors

“Real program code can have [errors], but there are more possibilities to verify things, also already at build time. Maybe that is a possibility to address this problem, that you may try to validate or check the configuration as part of building the software. [...] Everything that can be done as early as possible in the process, validation checking, helps to solve these problems.” — I_{11}

Avoiding Configuration Errors

"I don't think that my users [of the configuration editor] know what makes sense to configure and what not. So, it is my task to find out how to present the configuration options to the users in a way that it is as easy as possible to configure things." — I_{11}

Avoiding Configuration Errors

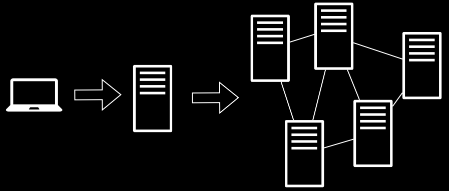
- explicit modeling of options
- favor value binding at build time
- simplify configuration

To avoid configuration errors, some companies implement some kind of validation for their configuration files. One practitioner stated: “Validation is part of the meta-configuration. [...] So, there is a kind of management model, that is, a configuration option is not only the name of the option, but also the data type, a validation, or where does this configuration value come from, so cross relations to other configuration values.”
To make best use of this validation, they apply a fail-fast strategy. They said: “Real program code can have [errors], but there are more possibilities to verify things, also already at build time. Maybe that is a possibility to address this problem, that you may try to validate or check the configuration as part of building the software. [...] Everything that can be done as early as possible in the process, validation checking, helps to solve these problems.”
Another strategy is to simplify the configuration. One practitioner that worked on a configuration editor said: “I don’t think that my users [of the configuration editor] know what makes sense to configure and what not. So, it is my task to find out how to present the configuration options to the users in a way that it is as easy as possible to configure things.”
To summarize, the practitioners suggested three main practices to avoid configuration errors: explicit modeling of options to make validation possible, favour value binding at build time over configuration at runtime to find errors fast and simplifying the configurations.

Dealing with Distributed Configuration

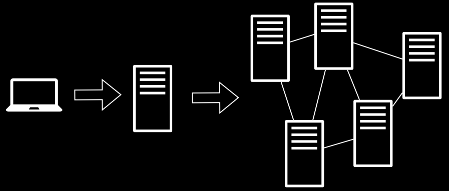
Dealing with Distributed Configuration

"A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things." — I_9



Dealing with Distributed Configuration

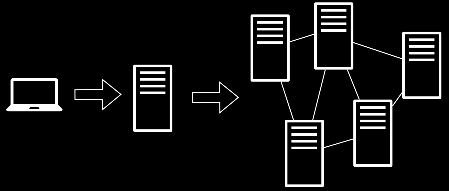
"A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things." — I$_9$



Dealing with Distributed Configuration

"A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things." — I_9

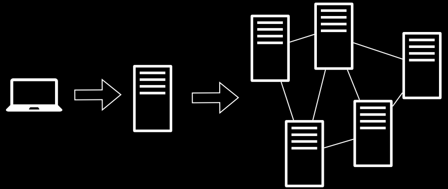
- dedicated platform teams



Dealing with Distributed Configuration

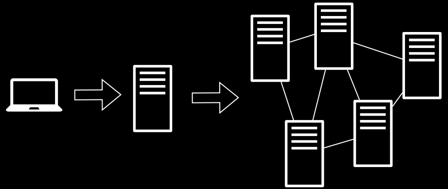
“A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things.” — I_9

- dedicated platform teams
- naming conventions



Dealing with Distributed Configuration

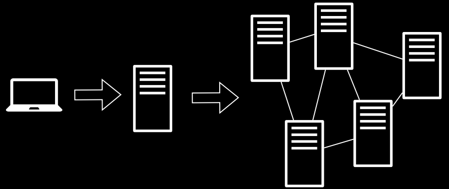
"A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things." — I_9



- dedicated platform teams
- naming conventions
- configuration files next to the code

Dealing with Distributed Configuration

“A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things.” — I_9



- dedicated platform teams
- naming conventions
- configuration files next to the code
- reviewing configuration in code reviews

To deal with distributed configurations, for example in microservice architectures, a lot of the companies had dedicated teams to configure the platforms. At one company a developer told us: “A sister team of ours mainly does this [Kubernetes configuration]. They provide the basis for us and provide the low level things.” This way, they had a clear separation of responsibilities and allowed the developers of individual services to focus on the configuration of their service without worrying about the integration to the infrastructure. Also some practitioners told us that there was a general naming convention for configuration options to make them consistent company-wide. They emphasized that it is important to have the configuration files next to the code they configure in the same repository to let it be tracked by the version control system. In addition, configuration files get reviewed in pull requests the same way that code gets reviewed.

Performance Engineering

Performance Engineering

“Performance is really difficult. What does performance mean? Response time of requests? From which requests? Resource consumption counts as well.” — I_8

Performance Engineering

“There is a team that does nothing else than performance testing. They have different setups at hand [...] and every new function becomes eventually part of a performance test.” — I_{11}

Performance Engineering

- Metrics for performance are not defined
- Performance tests for every new feature necessary

We have not seen many solutions to performance optimization despite its relevance. Performance engineers run stress tests and other related measurements, but with the main goal of finding good default values rather than optimized settings for individual customers. Interestingly, for the domain of performance engineering we could not find any best practices which is interesting since it seems to be an important topic in most of the companies. One reason could be that the term itself is—just like configuration—not clearly defined as stated by an interviewee who said: “Performance is really difficult. What means performance? Response time of requests? From which requests? Resource consumption counts as well.” While they would be interested in finding optimal configurations for individual customers or use cases, they only apply techniques like stress testing to find good default values. “There is a team that does nothing else than performance testing. They have different setups at hand [...] and every new function becomes eventually part of a performance test.”

Lessons Learned

Lessons Learned

- Configuration becomes increasingly important

“[Configuration] becomes ever more important, because we always thrive for writing less code and better reaching our goals via configuration.” — I_8

Lessons Learned

- Configuration becomes increasingly important
- Configuration lacks proper grammar and type system

"If I mix up some characters in my configuration, then there is no compiler that complains about that but everything goes through and later I have to find out that there is some character missing in the user name." — I_{11}

Lessons Learned

- Configuration becomes increasingly important
- Configuration lacks proper grammar and type system
- Configuration is a multi-person, multi-artifact, and multi-technology activity

“That reaches a certain level of complexity where I don’t know what happens if I turn a switch anymore. What happens with the configuration? Is it still active [...]? Or does it get overwritten by [...] other mechanisms?” — I$_{11}$

To summarize, I would like to present the lessons we have learned during our study. First, configuration becomes more and more important since: “[...] we always thrive for writing less code and better reaching our goals via configuration.” That is, instead of implementing everything themselves, developers tend to use third-party frameworks and tools which have to be configured.
Configuration often has no well-defined grammar and type system like source code has, which can lead to problems. An interviewee told us the following example: “If I mix up some characters in my configuration, then there is no compiler that complains about that but everything goes through and later I have to find out that there is some character missing in the user name.”
It is also a highly cross-cutting activity which affects almost all stakeholders of a software system, is defined in various artifacts which themselves are distributed in different environments. These influences are not always known to each stakeholder as one of the practitioners said. They said: “That reaches a certain level of complexity where I don’t know what happens if I turn a switch anymore. What happens with the configuration? Is it still active [...]? Or does it get overwritten by [...] other mechanisms?”

Additional material:

https://github.com/AI-4-SE/Dimensions-of-Software-Configuration

Pre-print:

https://sws.informatik.uni-leipzig.de/wp-content/uploads/2020/05/Configuration.pdf

Norbert Siegmund
@Norbsen

Nicolai Ruckel
@NicolaiRuckel

Janet Siegmund
@JanetSiegmund

This concludes my talk about the dimensions of software configuration. You can find our questionnaire, the transcripts, and additional material in our project repository at GitHub.
If you have any question, feel free to ask us during the interactive conversations or send us a message directly.
Thank you for your attention and I hope you enjoy the rest of the conference.