CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Action Research: Configuration complexity in software systems

Evaluating script automation as a tool for lowering configuration time and complexity when configuring network settings.

Master's Thesis in Computer Science and Engineering

Max Hagman, Oscar Palm

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

Action Research: Configuration complexity in software systems

Evaluating script automation as a tool for lowering configuration time and complexity when configuring network settings.

Max Hagman, Oscar Palm

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Action Research: Configuration complexity in software systems
Max Hagman, Oscar Palm

Supervisor: Miroslaw Staron, Department of Computer Science and Engineering
Examiner: Irum Inayat, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Max Hagman, Oscar Palm
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Previous research has shown methods and tools used for software configuration within many organisations to be unnecessarily time consuming, prone to errors, and complex. Research also shows that the complexity of tools used have a negative effect on the adoption of said tool within an organisation. This thesis investigated the effect of partially automated software configuration using scripting has on the complexity and configuration time of a system. The context of this thesis was a medium-sized software company, focusing on the configuration of their Software-as-a-Service network service. The research was performed using the action research methodology, to construct a tool, partially automating the configuration of software devices.
The created tool was evaluated on the difference in time, compared to the current solution, as well as the change in complexity. All the observed subjects needed slightly more, to moderately more time to complete a configuration using the created tool. The subjects also perceived the configuration process as less complex with the created tool. The actual complexity of using the created tool was also observed to be lower.
The results from this study shows how the complexity of a configuration can be improved by automation, even in cases where it is not possible to fully automate the task.

Keywords: Software Engineering, Software Configuration, Network Configuration, Network Management System, Automation, Software Complexity, Action Research, Computer, Networking, Configuration

Acknowledgements

First, we want to express gratitude to the people at the company where this thesis was performed; without your support and your tolerance to our questions and general pestering, this thesis would never have been completed.
We would also like to thank our lovely supervisor, **Miroslaw Staron** for helping us navigating through the treacherous swamp of research methodologies and their seemingly friendly and all-purpose descriptions, without your help, many more mistakes would have been made.

Max Hagman, Oscar Palm, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

The configuration of networks and network devices quickly becomes a necessity as networks change. These changes include network expansion, network devices being replaced, and the requirements for these devices changing. Configuring a network device is a time-consuming task [1], and configuring an entire network can take several days [2]. It has also been shown that a majority of incidents related to the availability of networks are caused by the misconfiguration of the network [2]. These facts combined make it highly desirable to improve the workflow regarding network configuration.

As software systems evolve over time, it is common for extra functionality to be added, and with that new configuration options [3]. There also exists a fear amongst developers for removing existing configuration options and functionality from their programs [4]. Combined, this means that the amount of configuration options frequently grows over time, and with it the complexity of the system. This increase in options and complexity makes it more difficult to understand and configure a system, but also increases the risk of introducing misconfigurations into the system.

This thesis was performed in collaboration with a company, using a Network Management System (NMS), to configure communication between their products and the wider internet. Using an NMS, configuration of a network becomes more akin to software configuration instead of network and computer configuration. As with software, over time, the configurations made within the NMS have become more complex.

As the needed use cases of NMS have grown, the possibility to define a network through software has appeared. The NMS used in the observed company exposes an API, allowing for Software Defined Networking (SDN).

By utilising SDN, a software engineer is able to treat the network as a software system, integrating it into existing software. This opens up several questions, which are interesting from a research perspective, for discussion. The question we choose to partially discuss in this thesis is the effect of using software automation for software configuration instead of manual software configuration.

Configuring a system with a large number of options is time-consuming and prone

to errors [3], errors that are often more severe than those caused by other issues, as found by Yin et al. [5]. These are issues that might lead to the system becoming unavailable or degrade its performance.

As found by Plonka and Tack [6], there are numerous gaps in research around the workflow of network configuration. Plonka and Tack show that in a local university network, system engineers frequently configured using a web-based GUI. The GUI automatically applies changes as soon as it is made. This forced engineers to frequently reconfigure the same device a second time within 10 minutes of the original configuration being made [6]. The same pattern is not established when configuring otherwise. This suggests that forcing system engineers to review and approve their changes before they are applied might be beneficial.

The purpose of this study was to understand network configuration through SDN within a company. Action research was used to reduce the time spent on configuration through third-party software, as well as lowering the complexity of performing these configurations. This study was performed inside a company, using system engineers to understand and evaluate the effects of the actions taken. More specifically, an already existing graphical user interface tool was evaluated and compared against a method using configuration files and a script written in Python. This aimed to increase the understanding of the effect that converting a sequential approach to a more declarative approach had on productivity. The impact was measured in the time a configuration took to perform and the complexity of the configuration process. The complexity was measured both through the perceived complexity by configuring users and using a mathematical model developed by Brown et al. [7].

This thesis aimed to broaden the understanding of the above-mentioned concepts by answering the following questions in relation to the following research problem:

RP 1: Configuring networks is too time-consuming

RQ 1: What effect does automation have on the time it takes to configure a new network?

RQ 2: How does the perceived complexity of configuring a network change when using automation?

The resulting method of working, as well as the process of creating the tool, can be used by practitioners to improve processes within their organisations, as well as by researchers to evaluate and understand the impact of different methods on the efficiency of configuring networks using software.

The action created can also be used to automatically duplicate a network when creating environments for testing and staging of new versions of software, or to restore the environment to a previous state if problems occur.

The remainder of the report is divided into chapters and structured as follows; Chapter 2 explains concepts needed to read and understand the rest of the thesis report.

Following this, Chapter 3 discusses other studies related to the thesis report and how this study aims to fill the gap left by these previous studies. Chapter 4 details differing scientific methods and explains how, when, and why these methods have been selected and used in the context of this study. In Chapter 5, the methods used, and results from the first cycle are presented. Chapter 6 describes the different threats to the validity of the results and what they have been used for. Chapter 7 concludes the report, drawing conclusions about the results and the discussion regarding those results.

2

Theory

In this chapter, concepts used in the study are explained in more detail. This chapter does not discuss the scientific methods used in the study, but the technical concepts needed to read and understand the report.

2.1 Software Configuration

Software configuration is the process of, without modifying the source code, changing the behaviour of a software through what is called configuration options [4]. These options can be configured in several ways. Either passed as parameters by the user, set in the program, an example being settings, passed through environment variables, configuration files, or through one of many other methods.

Software configuration options can be categorised depending on which phases of program execution it is possible to modify the options. Sayagh, Adams, and Petrillo describe four main categories of configuration [4] based on McConnell's description of the binding times of variables in programming [8];

- Pre-deployment
- Deployment/Installation
- Load time
- Execution time

Pre-deployment options are defined as being given a value before compilation. These options are usually used to decide if certain features are to be included or excluded in the compiled program. Changing these values requires recompilation of the entire program. McConnell exemplifies this kind of option with the use of a static variable as has been exemplified in (Figure 2.1) [8], but other methods also exist, such as C/C++'s `#ifdef`-statement allowing to pass options to the compiler [4].

Deployment/Installation options are given while installing the program, or deploying it in a system. These options can, for example, be used to change where and in what way the program should store data [4]. If the program is deployed using an

```
# Static variable, used to define background colour
BG_COLOR = 0x0fffe

window.background_color = BG_COLOR
```

Figure 2.1: Example in Python of a pre-deployment configuration option.

automation system such as Ansible or Puppet, these configuration options can also include configuring the environment the application is hosted in (what operating system to use, how it is connected to the internet) [4].

Load time options can be changed by restarting an application. These are commonly provided by arguments provided to the application when starting it or through variables saved in the system's environment.

Execution time options can be changed without restarting the program. These can, for example, be provided through a settings menu inside an application, or be configurations regarding the size of the screen, whether it is active or running in the background, and so on.

2.1.1 Software Configuration Complexity

Complexity, or more specifically computational complexity, is a measurement of how easy or difficult a certain task or problem is to complete [9]. Complexity in general is measured through counting the number of steps an algorithm needs to complete to compute a task [9].

Software configuration complexity differs in meaning, depending on the field discussed; it can be either a measure of the complexity of the program and its development, or the complexity of using the program and configuring it correctly. The focus of this report is on the second type of software configuration complexity: the complexity the program user experiences while creating new and updating old configurations.

For measuring and comparing the software configuration complexity, a method developed by Brown et al. [7] can be used, where the complexity is measured by counting:

1. The amount of context switches, meaning switching between different programs and different windows or views within this program.

2. The number of parameters being configured.

3. The amount of information needed to be remembered by the configuring party, and for how long.

These complexity measures are then compared against each other, and against counts of the same parameters using different tools, or performing different tasks.

This gives insights into how the complexity differs between the two tests and what creates the complexity.

Note that this is not a measure of a tool's or method's complexity, but rather the complexity of performing a configuration using said tool or method. For the data to be meaningful, a comparison needs to be made, either between different tools or methods used for the same configuration, or between different configurations using the same tool or method.

2.2 Network Management System

A Network Management System (NMS) is an application meant to automate the maintenance of a network, monitor the network, as well as manage the connection between devices on the network [10]. NMS works on the principle of a master client (the NMS), sending requests to its slaves (all other network components) [11]. These requests can be for gathering information, to be viewed in dashboards, or to apply a new configuration to said devices.

In the company, an NMS is used by system engineers to create new networks, assign different computers to these networks, and configure which path network traffic will take through the network, for one computer to access the internet.

An example of how an NMS is installed can be shown in (Figure 2.2). The filled-in lines represent the communication between devices in the network and with the internet, and the dotted lines represent which devices the NMS configures. The NMS does not automatically configure the network devices, but can rather be used to unify the configuration of several devices into one interface, as can be seen in (Figure 2.3), where all configurable devices are listed, and the user simply can select which device to configure and configure it through the interface, removing the need for system engineers or operators manually connecting to each device when configuring the network.

Using an NMS, such as the one presented in (Figure 2.3), configuring the network is done through clicking between devices with a computer mouse, and changing values in text fields. The NMS used as part of this study divides the configuration into a tree structure, where each node represents a logical object in the network. These objects include both the physical devices that make up the network, as well as the connections between devices, and the radio frequencies the devices are allowed to communicate over. Each object is populating individual pages, meaning to configure multiple objects, the engineer is required to open and close several pages, often navigating between two or more different pages to find the information required to fill out on another page.

The interface shown in (Figure 2.3) is not from the same NMS the company is using, but the dashboard structure, where each object is grouped into categories, and accessed through a file tree structure, and edited through individual pages,

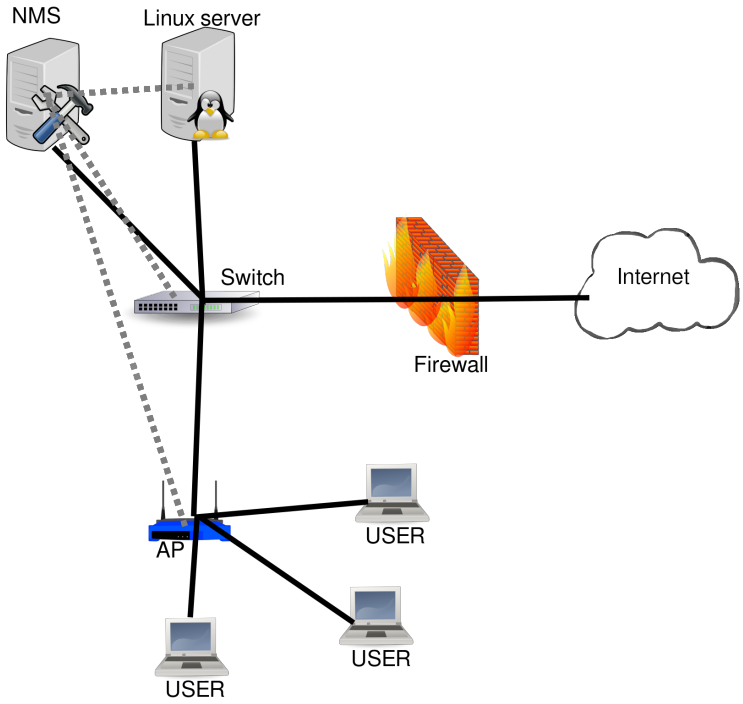

Figure 2.2: Example network with Linux server and NMS. The user devices and the server connect through a switch and a firewall to the wider internet. An NMS is set up in the network to manage and monitor all network devices, in this case, an access point and a switch, as well as the server.

is similar. It can be very time-consuming to configure even smaller networks [1], compared to using script automation [1]. The fact that an NMS, such as the one used within the company, takes time to use makes it a possible place to introduce an action to solve **RP1**. Further, since the navigation between several pages to access and edit information makes it a possible place to introduce an action to solve **RQ1-2**, and lower the perceived and actual complexity of configuring devices.

Since network devices such as routers run software designed to route and handle network traffic, the configuration of these devices can be viewed as a subtype of general software configuration. As such, an NMS is a tool allowing to remotely configure a software system.

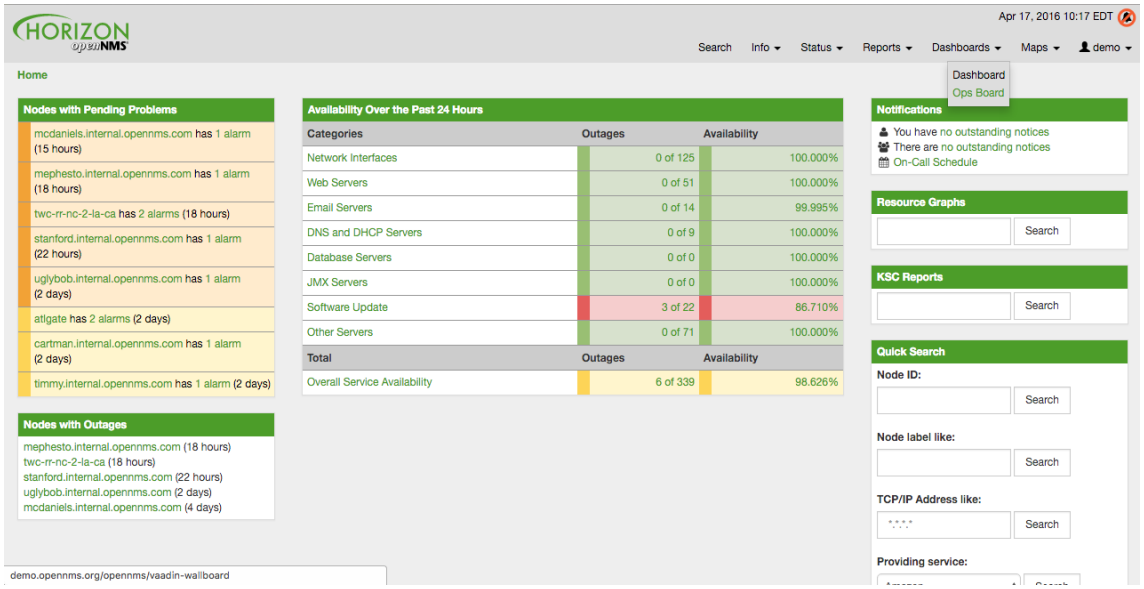

Figure 2.3: The graphical interface of an NMS (Taken from [12]).

2.3 Task complexity

Task complexity does not have a widely accepted definition [13], thus it is important to clarify what definition is used. In this report, the definition of task complexity is defined by Liu and Li [13]. Liu and Li define task complexity as a model, consisting of complexity contributing factors (CCF), task components and complexity dimensions. The model covers several views on task complexity, such as objective and subjective, but also structural, resource requirement and interaction.

A CCF is an aspect of task complexity and influences it, generally positively or negatively. Liu and Li define 29 CCFs but also state that for specific tasks or research, more or fewer may be relevant. Task components divide CCFs into categories based on what part of a task they are related to. There are 5 task components: *Goal/Output*, *Input*, *Process*, *Time* and *Presentation*.

CCFs also belong to a complexity dimension, which is a group of CCFs capturing a theme of complexity. Liu and Li define 10 complexity dimensions, which they state should be exhaustive and capture all aspects of task complexity. The 10 complexity dimensions are: *Action complexity*, *Ambiguity*, *Incongruity*, *Novelty*, *Relationship*, *Size*, *Temporal demand*, *Unreliability*, *Variability* and *Variety*. Similar to CCFs, Liu and Li state that for specific tasks or research, not all complexity dimensions may be relevant.

An example of how to measure the task complexity with the model is another of Liu and Li's published papers on comparing complexity in power plant main control rooms[14]. They utilised a 5-point Likert-type scale to measure three aspects from respondents: impact, complexity and frequency.

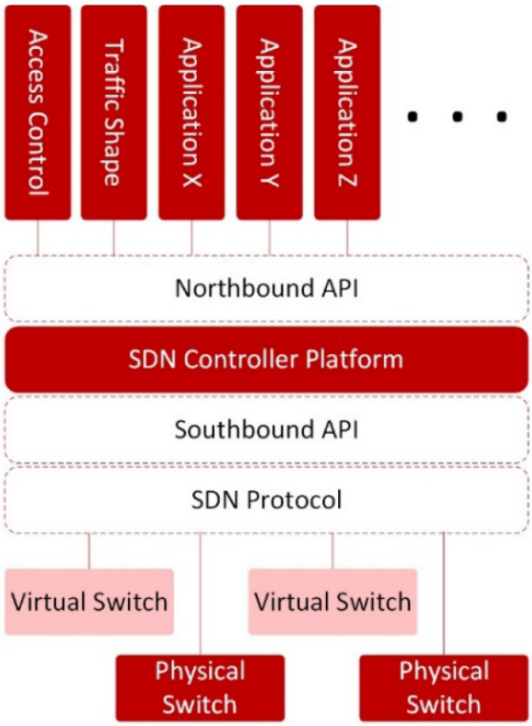

Figure 2.4: Simplified diagram showing SDN allowing applications to access and control network devices through an API. Taken from [15].

2.4 Software Defined Networking

Software Defined Networking (SDN) works by abstracting the physical devices into a vendor-independent programmable interface [15]. A simplified example of SDN usage can be seen in (Figure 2.4), where it can be seen how different applications configure a set of networking devices by sending requests to the northbound API.

An SDN is divided into different parts; the ones of interest for this thesis are the southbound and the northbound APIs. The **southbound** API allows the SDN controller to communicate with network devices and configure them [15]. The **northbound** API allows different applications to request changes in the network configuration [15]. This report regards everything after the northbound interface as a black box; a configuration change is sent to the northbound API of the SDN, and the network is configured.

One major benefit of using SDN is the ability to duplicate a network [16]. For software engineers, this duplication of networks is useful since it allows for the creation of identical environments for testing or when developing new features without the risk of introducing small inconsistencies, as when manually creating environments for testing.

3

Related Work

Since this thesis evaluated the effect of script automation on a configuration workflow through both the change in time the configuration takes, and the complexity of performing the task; the related work section has been split in two sections. The first section discusses works related to the effect of different workflow automations on configuration time. The second section discusses works related to the effect of automation techniques on the complexity of performing a configuration.

3.1 Configuration time

Previously, a study by Caicedo-Altamirano et al. [1] showed that the introduction of scripts for configurations of networks can decrease the time a configuration takes by 99%. It was, however, not clear whether the user configuring the network is familiar with the system or not, or what the scope of the configuration was. While the report indicates a clear decrease in the amount of time spent configuring networks by introducing python scripts, the report does not discuss the effect scripting have on complexity and perceived complexity of performing the configuration. The report by Caicedo-Altamirano et al. [1] employs a laboratory experiment to evaluate the effect of scripting. This study, in contrast, focused on the impact of scripting on network configuration times in a field environment.

The paper by Caicedo-Altamirano et al. [1] also has some issues regarding the presentation of their results, mainly due to a difference in what tasks are performed with and without running the script, as well as the previously mentioned issue regarding user familiarity with the task, as well as the network configuration.

Abbasi et al. [17] describes how using tools such as an API can be used to automate network management through the creation of scripts or programs. Using these scripts, they were able to decrease the amount of errors that appear while reconfiguring the network, and improve the reliability of the network [17]. Unlike the study by Caicedo-Altamirano et al., which focused on directly configuring network devices through scripts [1], Abbasi et al. shows that there is also an improvement in configuration time when not configuring network devices directly. Instead Abbasi et al. configured the network through an abstraction layer, using Software Defined Networking (SDN).

Liu et al. [2] in their case study present a tool made to automate network management and configuration. Through this automation, more than half of overall network incidents within the company disappeared, and they also measured a 93% reduction in the average processing time for updating the network [2]. While the studies by Caicedo-Altamirano et al. [1] present data from a laboratory setting, where the researchers have full control over every part of the system, Liu et al. [2] show the effects of automating a network, without abstraction through SDN, in an industrial setting.

For real time networks, Gutiérrez et al. [18] suggested a method of using existing technologies combined with the introduction of a configuration agent to automate the configuration of a network without external human input. While this method is proven to work well, it is not always possible to automate configurations in such a way. This study instead examined configurations of networks where the need for reconfiguration comes, not from the internal network, but instead from outside influence.

While these studies all discuss and improve on concepts within network configuration automation, they all assume the network is being reconfigured retroactively, after a change is perceived within the network; examples being increased load to certain machines, or the replacement of one or more network components. The goal of this study was instead to reconfigure networks after outside influence demands it, such as changing the desired entry point into the network.

3.2 Configuration complexity

The studies above have shown that using scripting and automation can greatly lower the time spent on configuration, but scripting can also have downsides. In a study by Voronkov, Martucci, and Lindskog, it is shown that many administrators preferred using graphical user interfaces (GUI) instead of command-line interfaces (CLI) and scripts because of the difficulty in learning to use them, and the lack of error preventing measures present in many graphical tools [19]. While there is a distinct difference between configuration of networks and configuration of firewalls, the insights regarding usability of CLI compared to GUI must still be regarded when researching network configuration methods.

If an organisation is already using a CLI, it can however not be assumed introducing any GUI is to be beneficial. In a study by Wei [20], it was noted for web based GUI tools, the adoption rate was linked to the complexity of the tool's interface. This shows the importance of lowering complexity, since all other benefits a new tool provides do not benefit the process if the tool remains unused. Supporting this, Davis et al. [21] found that perceived ease of use greatly affects the perceived usefulness of a tool, which in turn influences its use.

A study by Lee, Wong, and Kim [22], found that a large majority (4 out of 5) of modifications made to network configurations involved changing less than 15 lines of

configuration. It is unnecessary to automate all parts of automation, since the cost of automating and the loss of flexibility in configurations is too large compared to the benefits of configuration [22]. Further Lee, Wong, and Kim [22], mention using automation risks leading to a loss in knowledge about the system configured. The study however, does not mention what kinds of network configurations are worth automating. This thesis aims to partially fill this gap, by applying an automated workflow on one aspect of network configuration.

Brown and Hellerstein [23] provides a stronger reasoning for when automation is beneficial, noting depending on the time saved by using an automated process, an automation needs to be used a certain amount of times for the cost of implementation of the automation to be lower than its value. Similarly, automation of a task often adds extra complexity, and may result in more professions being needed if and when an automation fails, as well as to maintain the automation and maintain the infrastructure used by it [23]. This thesis, as part of **RQ2** analysed how automation affected the complexity of the whole configuration, not only the running of the tool; this was something noted but not further explored by Brown and Hellerstein [23].

3.3 Research Gap

The related work covers users' preferences between CLIs and GUIs, what is perceived easier to use. Ease of use is not complexity but could be argued to be a part of complexity or a result of low complexity. None of the studies explored complexity of performing configuration tasks.

Other parts of the related work focused on automation, though specifically adjusting the network configuration to internal changes. When external influence demands a change in configuration is not covered by the automation in the studies.

4

Research Methodology

This chapter aims to explain the different scientific methods used throughout the project, their purpose, and when and how they have been used.

4.1 Action research

Action research was the main research method used by the researchers in this thesis. Action research is performed in cycles, each cycle divided into five phases [24]. The goal of action research is to evaluate what is called actions, in the context of a company or organisation; with the result of a study being both a scientific paper, valuable to the researching community, but also the company learning and improving from the taken actions [24].

Staron [24] provides the following definition of actions: *An action is an activity done by or in collaboration with practitioners, which includes an intervention in the practice of the collaborating practitioners or their organisation.*

Diagnosing is the first phase in which the causes of the experienced problem are to be identified [24]. The findings of this phase builds the basis for the rest of the work the current cycle. It involves performing interviews, observations and more.

Action Planning involves utilizing the knowledge gained during the *diagnosing* phase to construct a plan on how to solve the identified issues [24].

Action Taking involves implementing the planned action and gathering data about what changed within the company [24].

Evaluation is the step where the teams decide whether the current cycle will be the last cycle, or if more cycles are needed. The data gathered during *action taking* is used to evaluate the success of the change [24].

Learning is the final phase of the cycle, where the changes made are presented to the company, making sure the knowledge gained is made long term [24].

4.1.1 Teams of action research

Three teams are involved when conducting action research [24]. There is an action team, which conducts the research and creates changes within the company, a reference team which the action team consults regarding their findings, and a management team which defines the scope and resources for the research.

The action team consists of the researchers, as well as practitioners within the company that together will plan an action, perform the action, and evaluate the effect of the action within the organisation [24].

The reference team consists of practitioners within the company, and exist to guide the action team [24]. At the end of each phase, the action team will present their findings for the reference team, which in turn decides if the result is satisfactory, and they can continue to the next phase, or if the artefacts created need to be reworked.

The management team consists of the management of the organisation and have two main responsibilities, one as managing what support the action team can gain access to, and one as deciding on the completeness of the action research project [24]. Since Action research works in cycles without a predefined end, it is up to management to decide when the research project finishes, either because of it having reached its goal, or because of a decline in interest from the management team or organisation as a whole.

4.1.2 Use of action research

Action research was used to understand the problems experienced by engineers in their work, and to create and evaluate changes to mitigate them. The cyclic and strong collaborative nature of action research made it a good fit for experimenting with the changes.

4.1.3 Context

The research was conducted at a medium size satellite communication company from Sweden. The company started providing a satellite broadband service with monitoring and interference detection, in addition to their already existing products, 1-2 years ago. The study was proposed to the company by the researchers involved, but financed wholly by the company. The service is developed by a group of 10-15 people, working as developers, testers, and technicians.

To develop the service, the team has taken inspiration from agile methodologies, dividing the work into epics containing different tasks, assigning each task and epic its own deadlines. However, instead of working in sprints the team members assign themselves tasks, with three meetings every week, where team members explain what they are doing and if they need help, or can help others.

The service is subscription based intended to be accompanied by their existing hardware product. To provide their service they use a Network Management System

(NMS) to configure the network connection between hardware the customer have bought from the company, and the open internet. The service is not one large software but rather a combination of several smaller components, some created by the company and some external programs.

4.2 Thematic Analysis

Thematic analysis is a six-step process created to analyse and understand the qualitative data gathered from interviews [25]. The results from thematic analysis are a list of well-defined themes, grounded in the combined interview results; these themes are meant to convert a set of qualitative data points into concrete problems and easily understood text while also making it more difficult to connect what is described to a specific individual.

The first steps of thematic analysis asks the researchers to thoroughly read the data, identify significant information and extract these as codes. These codes are then grouped into different themes, consisting of two or more codes describing similar information. The themes are then reviewed to ensure that they are internally consistent, and if necessary, several themes may be merged, split, or converted into sub themes of the more important themes. These themes are then given descriptive, short names, supported by a detailed description, and presented to possible readers.

4.2.1 Use of thematic analysis

The researchers conducted thematic analysis on the results from initial interviews with engineers as part of the diagnosing phase of action research. This was done to understand the common themes among different answers and to remove the connection between respondent and answer, anonymizing the data.

4.3 Goal Question Metric Framework

The Goal Question Metric framework is used to ensure coherence between what is being measured and why [26].

Goal represents the purpose of the research, explaining why the questions need to be asked. **Question** represents the questions needed to be answered for the researchers to know whether the goal has been reached. **Metrics** represent the quantitative data required to answer the question.

Each goal can, depending on its size, have one or more questions, and each question can have one or more metrics, and the metrics can also be used to answer one or more questions [26].

The process starts by defining what the goals are, after which each goal is broken down into several questions needed to be answered in order to understand whether

the goal has been reached or not. For each of these questions, one or more metrics are then defined to help answer them.

4.3.1 Use of Goal Question Metric framework

Goal Question Metrics was used to structure the results from diagnosing, with the goals being to solve the problems found through observing engineers' work and through analysing the themes created from thematic analysis of interviews. The questions and metrics was created to measure the success of reaching the goal.

4.4 Likert Scale

A Likert scale is used to measure attitudes and consists of a group of Likert-type responses [27]. It allows the use of parametric statistical methods, despite Likert-type responses being an ordinal. A Likert-type response is typically 5 points, where common values are "Strongly Disagree", "Disagree", "Undecided", "Agree", and "Strongly Agree".

4.4.1 Use of task complexity for creating Likert type items

Every Likert type item was created from the description of each CCF given by Liu and Li[13]. This means that each item corresponded to a specific CCF.

However, not all CCFs was evaluated. Almost all CCFs of the **Goal/Output** dimension was not considered as the given goal and desired output was the same for the evaluation. The other excluded CCFs was those of the objective nature, as described by Liu and Li[13], due to the survey measuring subjective complexity.

For an overview of the relevant CCFs and their relationship to complexity, complexity dimensions and task components, see (Table 4.1). The CCFs were grouped by their task component and lists the complexity dimensions they belong to. The rightmost column shows how the CCF affects complexity.

4.4.2 Use of Likert scale

Likert scales was used in a survey to measure the perceived complexity of using the UNC during action taking in the first cycle. The survey was answered by the subjects of observation directly after first using the UNC.

Since the number of respondents were much fewer than the complexity comparison study performed by Liu and Li[14], where they had 69 respondents, the responses was discussed individually instead of utilizing common quantitative techniques such as factor analysis.

Task component	Complexity contributing factor	Complexity dimension	Relation to complexity
Goal/Output	Clarity	Ambiguity	Decreases
Input	Clarity	Ambiguity	Decreases
	Inaccuracy	Unreliability	Increases
	Unstructured guidance	Ambiguity	Increases
	Mismatch	Incongruity	Decreases
	Non-routine events	Novelty	Increases
Process	Repetitiveness	Novelty	Decreases
	Cognitive requirements by action	Action complexity	Increases
	Clarity	Ambiguity	Decreases
Presentation	Heterogeneity	Incongruity	Increases

Table 4.1: An overview of complexity contributing factors used for the survey along with their complexity dimension, task component and their effect on complexity. (Taken from *Liu and Li*[13]).

4.4.3 Configuration complexity model

As previously shown in Section 2.1.1 , the complexity of a configuration can be modelled and measured by measuring the execution complexity, the parameter complexity, and the memory complexity.

Different types of configuration actions were measured, this included adding a parameter, modifying a parameter, switching to a new context, adapting a value into a new format, and using the same parameter in different contexts. By measuring the amount of configuration actions performed, as well as what type each action is of, it can be better understood what makes performing a task using the specific tool, complex.

5

First Cycle

The methodology used and results from the first cycle are presented and structured chronologically.

5.1 Team structure

The team compositions for the cycle are presented in (Table 5.1). The system engineer part of the action team had one year of experience working with the specific system, and helped by providing feedback during development and in creating requirements and a scope for the action. The system engineer did not however have previous experience in development. The software developer had worked for more than one year within the company, developing internal services used by system engineers.

The engineers in the reference team had several years of experience working with the system. The senior system engineer worked on maintaining the software system, while the senior software developer worked on creating tools for automation, and to help manage and maintain the system. Together the reference team helped by providing guidance and feedback related to what aspects of the workflow to focus on, and what was important to the company.

The management team consisted of the same senior system engineer as is part of the reference team, along with a chief product officer, responsible for the system and the team working on it. The chief product officer had been responsible for the system for several years, and had overseen the creation of the subscription-based service this research project was focusing on.

Action team	2 Researchers, 1 Software Developer, 1 System Engineer.
Reference team	1 Senior software engineer, 1 Senior System Engineer
Management team	1 Senior System Engineer, 1 Chief Product Officer

Table 5.1: Presentation of the team composition of the cycle specifying the members' role in the organisation.

5.2 Diagnosing

The first step of diagnosing was to formulate a theory, explaining what wass perceived to be the relevant problem, why this wass a problem, as well as what wass perceived to have caused the problem. The theory can be found in (Table 5.2).

Theory	Configuring networks using the NMS is time-consuming and confusing.
Rationale	This theory describes our understanding of how a user interacts with the NMS.
Definition	A configuration error is a missing or set value for the configuration that causes unintended behaviour.
Theoretical Model	The NMS interfaces used to configure the different system components consist of nested menus and pop-up windows. This, combined with a large amount of configuration options needed for each network component, makes it difficult for system engineers to update or create new networks. Not all components can be copied and none can be moved to other parts of the network, resulting in many components needing to be created from scratch. One value is prefilled from other values. The process can be improved by allowing components to be moved, copy old components, have templates for components and perform domain specific calculations for the user to prefill more values.

Table 5.2: The theory as defined for the diagnosing phase of the first cycle.

The defined theory was used as a basis to prepare observation protocols and questions for interviews. Protocols for the interviews and the observations are presented in A.1 and A.2, respectively.

Overall, two interviews were held with different engineers at the company, and three different observations of one engineer at work was performed. All interviews were recorded and then transcribed. The observations were screen-recorded to be able to review the actions taken after the observation.

To better understand the data gathered through interviews, thematic analysis in accordance with Ahmed et al. [25] was performed. Afterwards, the themes created were compared against the data gathered from observations.

5.2.1 Diagnosis result

The problem to be addressed in the cycle was:

RP1 Configuring networks is time-consuming

RQ1: What effect does automation have on the time it takes to configure a new network?

RQ2: How does the perceived complexity of configuring a new network change when using automation?

- How much does the usage of a script for automation affect the complexity of the configuration process?

5.3 Action planning

Following the diagnosing phase, the information gathered and presented to the reference team was used to describe the goal of the first cycle. From this goal, a GQM was created to better understand what is needed to measure the success of reaching said goal.

5.3.1 Goal Question Metrics

The research questions, as well as the goal of the cycle was structured in accordance with the Goal Question Metric (GQM) framework.

The goal and questions were based on the results gathered from diagnosing, while metric **M2** and **M3** were based on research presented by Brown et al. [7], in which they discuss various methods of measuring the quality of configuration systems. Metric **M1** for question **Q1**, measured the total time to perform the configuration from start, to a complete configuration. Excluded from the total time were toilet breaks that cause the subject to not actively work towards a complete configuration.

Goal 1: Reduce the time spent on performing the task ”Add network” (G1).
Q1: What effect does automation have on the time it takes to complete the task ”Add network”?
M1: Total time spent on task
Q2: How does the complexity of configuring a new network change when using script automation?
M2: Complexity measurement framework presented in Section 4.4.3 .
M3: Perceived complexity of performing task, rated using a Likert scale [27].

5.3.2 Design of UNC

The section covers the requirements and design of UNC.

5.3.2.1 Defining the tool

The company also noted the importance of any tool created being easy to use, as to increase the likelihood of employees adopting the tool into their workflow. While the tool created here is not web-based, it is reasonable to assume similar effects on other computer-based interfaces.

The system engineers also use computers with different operating systems, meaning

the tool needed to work on most commonly available computer systems without the need for a complex installation process. The full list of requirements elicited from discussions within the action team can be found in (Table 5.3).

Requirement	Evaluation
Works on all operating systems used within the systems team	Run the script on all used operating systems
Runs only on the Python3 default package library	Run the script on all used operating systems without installing external packages
Easy to distribute new versions, and install on new computers Easy to modify	The script is contained within one file Over 90% of functions are documented, and the cognitive complexity of every function is lower than 15
Linting of user input	Common mistakes, such as using comma instead of dot, are corrected without user input.
Validation	User is given multiple chances to review input and correct mistakes before a configuration is applied.

Table 5.3: List of elicited requirements for the script, followed by description of how the requirement is evaluated..

To account for the limitations presented in (Table 5.3), the following decisions and controls were made. The action was decided to be a script written in Python3, automating one of the most time-consuming and repetitive parts of the configuration process. We call this script Unified Network Configurator (UNC). To further lower the complexity of the setup and use of the UNC, only libraries included in the default installation of Python 3 on Windows, as well as macOS and a standard GNU/Linux distribution (Debian 13), were used.

For reading the data needing to be updated, an ini-file was used, where users input the new values used for configuring different parts of the system. This allowed the same configuration to be used at a later time, and for users to modify only certain parts of the same file when configuring different components similarly.

A flowchart was created to visualize to the reference team the proposed flow and to agree on where users need to interact with the UNC as well as what information they need to provide. The flowchart can be seen in (Figure 5.1). It can be seen that the script should

1. Read and validate a configuration (ini) file.

2. Authorize itself towards the NMS.

3. Find the network to modify.

4. Search for relevant linecards, and gather information about the linecards.

5. Apply the changes to the network devices, and present the new configuration to the user.

6. Update the devices' configuration through the NMS.

As can be seen in the flowchart ((Figure 5.1)), it was decided that the user was to be required to validate the changes made after steps *Validate ini file*, *Search for network*, and after the final *Get linecard data*. These validation steps gave the system engineer opportunity to see what devices was to be configured, and how the configuration options would change after the configuration. By providing information about changes before they were applied, it was possible to see and correct mistakes before they were applied.

5.3.2.2 Development of the tool

To make the distribution of the UNC between users simple, a script was created to combine all project files into one script file, we call this the PPS-creator (Portable Python Script creator). This allowed for splitting the project into different files when developing, lowering development complexity [28] and time while also keeping the ease of installation and use low.

The PPS-creator was configured to run on every commit pushed to the main branch of the distributed version control system used by the company. A flowchart for this script can be found in (Figure 5.2). As seen in the flowchart, the PPS-creator works by:

1. Finding all relevant python files.

2. Separating the imports from the functional part of the file.

3. Adding all functional parts into one file, with the main file last.

4. Removing duplicate imports, and appending the imports to the beginning of the file.

By moving the imports to only be in the beginning of the outputted script, the outputted script remains structured similar to the individual files that was combined, and by always adding the functional elements of the main file to the end of the outputted script, the entrypoint of the script remains at the end of the file, and the entrypoint of the script is ensured to be located at the same location within the script, making it easy to make changes to the outputted script directly if needed.

A dependency graph can be found in (Figure 5.3). The graph shows all created modules that are part of the script and which modules each module depends on. Each module is described below.

The Logger module allowed for user control of the verbosity of the script, messages

were divided into the categories error, warning, info, and debug, and depending of the needs of a user, a verbosity level could be provided, with all messages of the given verbosity level, and the more important verbosity types being logged. For example, a user selecting the verbosity level "warning", would be shown warning messages and error messages, while info and debug would not be shown. Further these messages was also stored in a log-file, for traceability of program usage.

The Reader module performed two different tasks; it read data from an ini-file, storing the changes to be applied to the system. It also wrote the current configuration for the parts of the system affected by a change before the configuration was applied. This saving of the old configuration allowed both for easy restoration in case of failure, but also allowed duplication of the environment for creating a test environment.

The Models module contained representations of different configurable objects in the environment, as well as classes for comparing different versions of the same object, and validating objects. To avoid the Tower of Babel anti-pattern, where the same data is represented in different ways internally, when communicating between parts of a system, or in different parts of the system [29], we decided to not represent the different system object configurations as different classes, but instead use the python typed dict.

Validation held different helper functions used to validate the typed dictionaries. These functions were used to validate if a field had a value of correct type, but also if that value was of the correct format and correct range. An example of this can be found in Chapter B.

The Clients module mainly consisted of a class `NmsClient`, which communicated to the NMS API,the northbound interface of the SDN. `NmsClient` was used to read the current configuration to be used for comparison and validation of the proposed changes, and to apply the new configuration values to the SDN. To allow for testing in different ways, the client could be used either with a `HttpsClient`, sending requests over the internet to a given IP address, or with a `DryRunClient`, which was an extension of the `HttpsClient` able to read data from a given web server, but logging the requests meant to apply data to a web server, more about this can be read in Chapter B.

The Configuration template file the users got included all possible parameters the user was able to configure. This was decided to keep the configuration parameters consistent with the existing tool. Hiding configuration parameters might have impacted the perceived complexity, making it more unclear whether the change in complexity was caused by the tool, or the design of the template. Instead the researchers decided that the design of configuration templates was relevant to cover in a future cycle.

5.3.3 Evaluation Design

Covers the planning and design of the evaluation.

5.3.3.1 Survey instrument

A survey was created to compare the complexity of UNC with the current workflow. The survey consisted of 8 Likert type items that each corresponded to a CCF. The 8 CCFs that was selected were:

1. Input Clarity
2. Inaccuracy
3. Unstructured guidance
4. Mismatch
5. Repetitiveness
6. Cognitive requirements by action
7. Process Clarity
8. Heterogeneity

The CCFs and their corresponding questions can be found in (Table 5.4).

As an objective measurement was performed with the complexity measurement framework, the survey was built on a subjective view on complexity. Any CCFs that measured an objective aspect of complexity, such as the amount of outputs was not included in the survey, as this was already covered in the complexity measurement framework and also not relevant for measuring the perceived complexity.

The Likert type items are listed in (Table 5.4) along with the CCF that they were designed from. When constructing the Likert type items the name and description of the CCF was used to create a statement. If necessary, vague or general words, such as "task", "action" and "goal", were replaced with context specific ones to increase the clarity of the statement.

The items were scaled on 7 points where the respondents were asked to select the option that they believed best fit their experience. As each item compares the respondents experience using both tools, the further away from the middle in the scale, the stronger the statement fits the tool used when compared to the other tool. Each point in the scale can be represented as: -3 -2 -1 0 1 2 3, where the negative numbers represent a stronger fit for the NMS, and the positive numbers represent a stronger fit for UNC. An answer of 0 is an equal fit.

Complexity contributing factor	Statement
Input Clarity	It is clear what information is required to achieve the desired configuration
Inaccuracy	The instructions given during configuration are accurate
Unstructured guidance	Guidance is unstructured (checklists, step-by-step, etc)
Mismatch	The presented information is easy to understand
Repetitiveness	Few repetitive actions are needed when configuring
Cognitive requirements by action	It requires a lot of attention to perform the configuraton
Process Clarity	It is clear how to achieve the configuration
Heterogeneity	Information is presented in an inconsistent way

Table 5.4: The questions used in the complexity evaluation survey and the CCF they were designed to measure.

5.3.3.2 Collection of Data

To collect the data to later be analysed and evaluated, system engineers was observed, timed, and recorded while performing the task affected by the action. This task was performed twice, once using the current tools and once using the UNC.

Since the configuration of the network changed after configuring the network, and to avoid interfering with the company's network; a snapshot of the network was stored and used inside a test environment. This test environment also allowed for restoring to the same network state before collecting the data of the UNC and the current tools and workflow.

The task was to setup a new network according to a specification. The specification used real data but had not been used before. This ensured that all steps of the task needed to be performed. A list of all parameters included in the specification can be found in (Table 5.5). The specification included 2 devices, each of which had values for all parameters listed in (Table 5.5). The carrier-parameter contains an ID, identifying which device was supposed to be included in the created network, and the label includes information about the devices' role in the network, such as if it transmits or receives data.

Further, a survey with Likert-type question responses was created for the observed person to answer. The answering of this survey was also observed, and the respondents were asked to explain why they selected their options as their answers.

The time measured during the performance of the task was used for metric **M1**, time taken; and the recordings was used with the complexity measurement framework [7] to measure metric **M2**, complexity of configuration. Memory related parameters of the complexity framework were omitted due to a lack of tools to properly measure

Carrier	Bandwidth
Label	Power
Modulation details	Output Back Off
Bitrate	Up Frequency
Spectral Efficiency	Down Frequency
Symbol rate	Up Start
Spacing factor	Up Stop
Transmitter Effective Isotropically Radiated Power	

Table 5.5: All parameters present in the specification used by participants. Values have been removed.

them.

Answers to the survey was analysed, in combination with comments gathered during the observation of respondents answering the survey, to measure metric **M3**, perceived complexity.

5.3.3.3 Experience levels with tools

Since the engineers all had different backgrounds and experience with the tools, no comparison was made between them. Instead, the experience level was presented for each engineer, and the differences will be contrasted to understand how the previous experience influences the result.

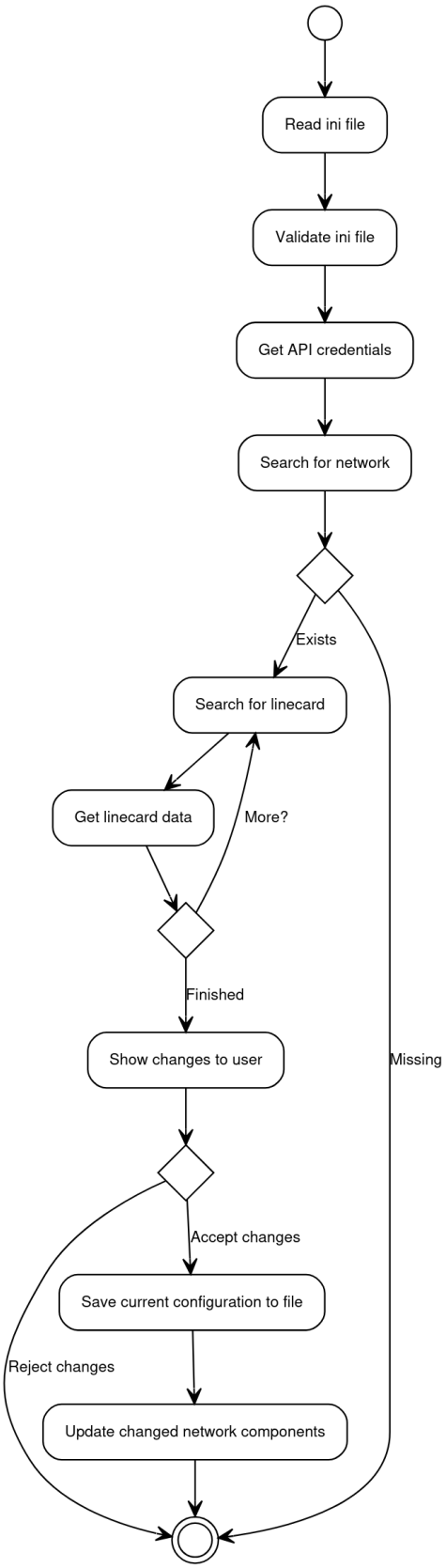

Figure 5.1: Flowchart defining what the UNC does, and in what order. As shown, the only required device for performing a network configuration is the network, while linecards can be added as necessary.

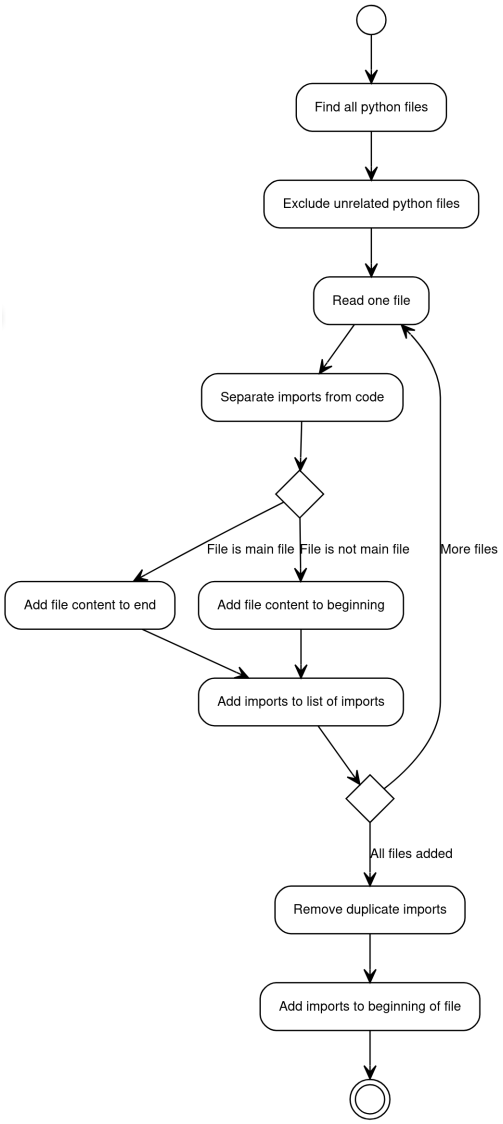

Figure 5.2: Flowchart defining what the PPS-creator does, and in what order. Aside from combining several python files into one script, it also removes duplicate imports and ensures the entrypoint of the script, as well as the main method is at the end of the script.

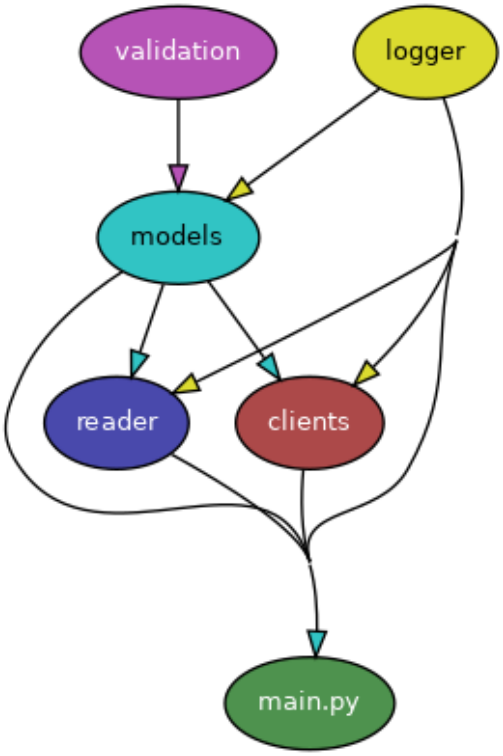

Figure 5.3: Dependency graph of the script. The arrows represent which modules a given module is used within. As an example, the models module depend on the validation module.

5.4 Action taking

Data about the impact of the action taking was collected from four engineers, with a total of six engineers involved in the action taking as part of the action. The two engineers not used for data collection were involved in the research project as part of the reference team and the action team respectively.

During data collection of the first engineer, a few issues were discovered in the action, these issues were caused because of changes made to the test network during the company move. How this was solved, and the potential impact is discussed in Chapter 6.

5.4.1 Collecting data

The engineers were invited to individual meetings, where researchers presented a task to be performed first using UNC, and then once again using the current workflow with the NMS. The UNC was briefly introduced and explained before the engineer started performing the task. When questions arose, the researchers noted what was being asked, and answered.

The subjects screens were recorded and observed by the researchers while they were configuring the network. After the subjects was done the researchers followed up with a Likert-type survey, comparing and evaluating the two ways of working. The researchers noted comments raised by the engineers during tool execution and survey answering, the purpose of these notes was to understand why something went wrong, to understand what information, and what type of information, was needed in later phases of the action cycle, primarily during learning.

5.4.2 Extracting data

For metrics **M1** and **M2**, the researchers collected screen recordings of various length. To enable researchers using the data, each video was analysed manually, and the data points required was extracted from the recordings. The parameters extracted, and their usage can be seen in (Table 5.6).

To ease the collection of parameters, a web based tool was created, where counts of values were entered, stored, and fetched. This made it more efficient to analyse the video data and provided a structured output making comparison between different tools and people easy. This also made it possible for multiple people to analyse the same data and compare to lower the risk of counting errors.

When extracting the data, it was noticed than manual extraction of the memory parameters were more time-consuming than at first thought. Since the manual measurement of amount of information kept in mind at one time, and length of time information was kept before being used turned out to be subject to great interpretation and imprecise measurement it was decided to not use the parameters regarding memory size, memory depth, and memory latency. The parameters are

still visible and described in (Table 5.6), but have not been collected.

Metric	Name	Description
M1	Time spent	Total time actively performing the configuration or understanding the tools.
M2	NumActions	Total amount of actions performed, this includes things such as opening program, providing information, reading information, and more.
M2	ContextSwitchSum	A context switch occurs when the engineer switches from configuring one part of a system to another part, such as when switching from configuring one computer in the network to another computer.
M2	ParamCount	Total amount of configuration parameters changed.
M2	ParamUseCount	Number of times a parameter is used.
M2	ParamCrossContext	Number of times the same parameter is used in different contexts.
M2	ParamAdaptCount	Number of times the format of a parameter is adapted, for example switching from writing size in MB to size in KB.
M2	ParamSourceScore	Each parameter is graded from 0 to 6 depending on the ease of understanding what its value should be, this count is the aggregation of all parameters' score.
M2	MemSizeMax	A measure of how many parameters are in the
M2	MemSizeAvg	user's memory when an action is performed. Each time a parameter is seen it is added, and when it is used it is removed.
M2	MemDepthMax	When memory is viewed as a LIFO-stack, the
M2	MemDepthAvg	amount of parameters on the stack "above" the needed parameter.
M2	MemLatMax	The amount of actions performed between the
M2	MemLatAvg	storing of a parameter value, and the usage of it.

Table 5.6: Parameters extracted from the recordings, and corresponding metrics.

5.5 Evaluation

This section is divided into five subsections, one subsection discussing the overall results and giving an overview of the data used, and one subsection for each engineer analysed (Subjects 229, 411, 625, 697). In these sections the reason for the results gotten is discussed.

5.5.1 Data overview

Collected data from the survey and the execution of the evaluation task is presented in the following sections. The data is gathered from four subjects and only once as there is one task, resulting in a restricted options for statistical inference.

One subject did not fully complete the configuration task with the NMS. Most of the configuration was successfully performed but due to missing a step they were unable to apply the configuration correctly. It is hard to say exactly the effect this has on the results. It is unknown exactly how much more time the configuration would take to complete as well as how the results of the measured parameters would change. The survey responses should still portray the subjective view on complexity of the subject as intended although there is a risk of bias towards the new tool due to completing the configuration successfully with it. In the data presented here on forth what we collected is presented as is and it is worth keeping in mind when viewing the results.

5.5.1.1 Perceived Complexity

An overview of the survey results showing which tool the respondents thought was least complex is presented for each CCF along with the associated category in (Table 5.7). The scale of the answers was disregarded due to it being different for each subject. As such the results should be interpreted as likelyhood of an improvement or a regression rather than how much of an improvement there is.

Overall when looking at every category there was a clear difference in the amount of responses for each tool, 6 and 16 for the NMS and UNC respectively. Indicating that there was likely an overall complexity improvement. In contrast ”presentation” and its only CCF, *heterogeneity*, was stagnant in its complexity and ”input” was showing a slight improvement overall, with a 1 and 3 response difference respectively. ”Process” showed the greatest improvement with a difference of 6.

Unstructured guidance was the CCF where regression was most likely while *mismatch* and *repetitiveness* were the most likely to see an improvement.

An overview of every subject’s least complex tool for each complexity dimension can be found in (Table 5.8). While all people surveyed favoured the UNC over the current tool for most complexity dimensions, some dimensions had varied perceptions by different surveyed subjects. This difference in perception can be explained by a difference in previous knowledge of the current tool, as well as different people

Category	CCF	NMS	UNC
Input	Clarity	0	2
	Inaccuracy	2	1
	Unstructured guidance	3	1
	Mismatch	0	4
	Total	5	8
Process	Repetitiveness	0	4
	Cognitive requirements by action	1	1
	Clarity	0	2
	Total	1	7
Presentation	Heterogeneity	0	1
All categories	Total	6	16

Table 5.7: The amount of responses in favour of each tool, disregarding the degree. 0 value answers are not part of the amount presented.

Dimension	229	411	625	697
Ambiguity	UNC	UNC	UNC	UNC
Unreliability	UNC	UNC	Same	NMS
Incongruity	UNC	UNC	UNC	UNC
Novelty	UNC	UNC	UNC	UNC
Action complexity	Same	NMS	Same	UNC

Table 5.8: The lower complexity tool in each complexity dimension, as perceived by each subject.

solving the given task in different ways.

Notably, the action complexity dimension did not show either tool as being more complex, with every subject answering either the current tool was slightly more complex, the UNC was slightly more complex, or both tools were similarly complex.

5.5.1.2 Complexity framework

As can be seen in (Figure 5.4); for all examined users except one, the total amount of actions performed was lower using UNC compared to using the current tool. Subject 625 showed a large amount of the actions performed using the current tool were not necessary. During observation it was noted several tasks were repeated multiple times, as the tool did not allow for saving a non-complete configuration, and did not allow accessing of information from different parts of the system at the same time as editing.

The mean values of each parameter can be found in (Table 5.9), as can be seen, using the UNC, most users configured fewer parameters compared to when using the current tool (24.35% fewer parameters modified), and modified these parameters fewer times compared to using the current tool (26.18% fewer uses of the parameters).

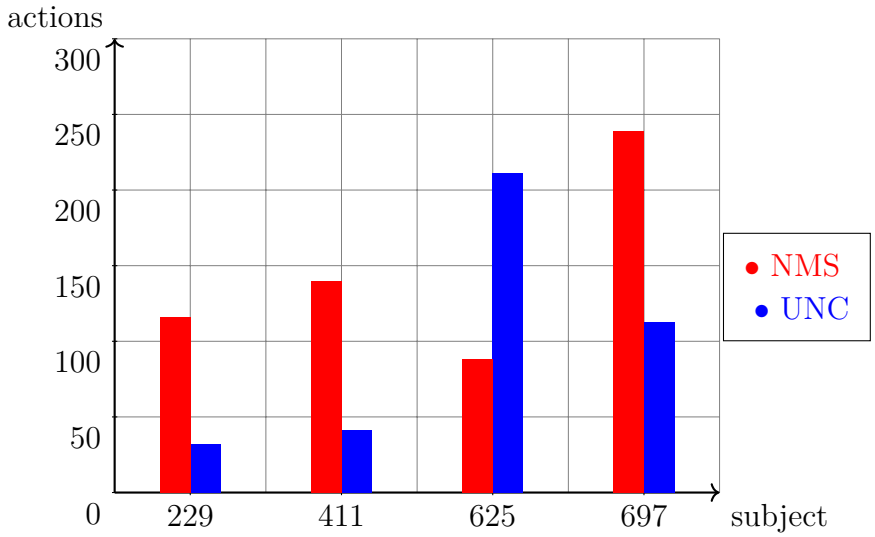

Figure 5.4: Total number of actions performed by a user using a specific tool. For most users, a significant decrease in total amount of actions can be seen using the UNC.

Using the existing tool however, resulted in switching between contexts less frequently on average, with 17.60% more context switches while using the UNC. When looking at the average amount of context switches using the UNC and the current tool, the average user switched context 55 times with the new tool, and 83 times with the old tool. This discrepancy is caused by one user performing 290.63% more context switches using the UNC, this will be further discussed in Section 5.5.4 .

The average cross context parameters used being lowered with the UNC was caused by the current tool forcing the configuring subject to recreate entire configurations for certain devices, while the UNC instead allows modifications of only the needed fields.

The amount of parameters adapted before being submitted is not a reflection of the tool, but rather how different subjects used the tools. In some cases the subject observed chose to themselves calculate the value a parameter should have when using the UNC, while other users copied that same value from other sources, and using the current tool similar scenarios also affected the result.

5.5.1.3 Time taken

The time taken to perform the configuration with the UNC as well as the NMS is presented in (Table 5.10). The time taken to perform the configuration with UNC was longer than with the NMS by 5%-378%. Comparing median increase in time and average increase in time, we see that while the average increase was 112.29%, the median was much lower, at 33.29%, suggesting a significant variation in time spent configuring with the UNC.

The subject with most previous experience with tools similar to the UNC, sub-

	Mean		User reduction (%)	
Name	NMS	UNC	Mean	Median
action	145,75	99,25	14.02%	61.72%
contextSwitch	83	55	-17.60%	68.44%
paramCount	34,5	26,25	24.35%	31.73%
paramUseCount	46,25	34,25	26.18%	39.54%
paramCrossContext	2,5	2	31.25%	25%
paramAdaptCount	1	1	0%	0%

Table 5.9: The averages of each parameter relating to metric M2 is shown on the left (Average) for both tools. To the right (User reduction) is the average and median of the reduction of each parameter per user. Negative values are an increase and positive are a decrease when comparing UNC to the NMS.

Name	Time taken with NMS	Time taken with UNC	Increase in minutes	Increase in percent
229	00:19:18	00:21:12	00:01:54	009.84%
411	00:13:15	00:20:45	00:07:30	056.60%
625	00:07:38	00:36:28	00:28:50	377.73%
697	00:47:07	00:49:28	00:02:21	004.99%
Average	00:22:40	00:35:34	00:10:09	112.29%
Median	00:16:16	00:28:50	00:04:56	033.22%

Table 5.10: The time taken to perform the configuration using the NMS and the UNC.

ject 229, completed the configuration with the smallest difference in time, with a difference of 1:54 minutes, which was an increase of 9.84%. Similarly, the subject with least experience with the current tool also experienced a comparatively small increase in time spent configuring, with 4.99%, or 2:21 minutes, more time spent configuring with the UNC.

5.5.2 Subject 229

This person had a significant amount of experience using the current tool. The person also had experience with tools similar to the UNC.

5.5.2.1 Perceived Complexity

This person pointed out that the templates for configuration files is overwhelming, the template given to users include every possible configuration option that can be provided, while a user performing a given task will only use a subset of them, providing different templates for different tasks, hiding unnecessary and uncommon configuration options would make the configuration process less complex.

The results from the survey can be seen in (Table 5.11), and be explored further in the following paragraphs. For *input clarity*, the template providing all possible

options that can be set instead of providing the options needed for a task reduces the input clarity. Since the current tool separates inputs into different objects, disallowing the user from inspecting the entire configuration at once, the two tools achieve a similar input clarity.

For *unstructured guidance*, the list format of the UNC provided a more clear overview of the configuration, compared to the split views where parts of the configuration were viewed separately in the current tool.

For *process clarity*, both tools were unclear in what information they require the user to fill in, but for different reasons. The current tool was unclear since the information was filled out as parts of different objects, providing a subset of options at a time, while the UNC instead provided all options as of the same importance, making it unclear what information was needed and what was optional. The person did not provide further reason as to why both tools had a similar process clarity.

Regarding the **ambiguity** dimension, the factors contributing to our understanding of it had a combined leaning towards the UNC being more ambiguous, with *unstructured guidance* suggesting the UNC being slightly more complex and *input clarity* as well as *process clarity* suggesting comparable complexity between the two tools.

Regarding *inaccuracy*, the current tool provided feedback to the user directly when providing a configuration option, while the UNC provided feedback after all configuration options were set. This separation in time between the user submitting a value and receiving feedback made it less clear what was incorrect. Since inaccuracy is the sole CCF affecting the **unreliability** dimension, it also suggests a lower unreliability for the current tool, compared to the UNC, for this person.

When discussing *mismatch*, access to the full list of configuration options in the UNC made it easier to understand what information needed to be set, when compared to the object structure provided in the current tool.

For *heterogeneity*, the current tool provided access to all objects part of the system, while the UNC specified a network, and a smaller number of devices per file, this separation made it more clear what needed to be changed, hiding the rest of the system from the user.

For the **incongruity** complexity dimension, both contributing factors suggests a lower incongruity complexity for the UNC, with the mismatch CCF suggesting the UNC had lower complexity compared to the current tool, and the heterogeneity CCF suggesting a slightly higher complexity for the current tool.

Regarding *repetitiveness*, the design of the current tool made it necessary for the user to perform more repetitive tasks, especially when moving devices between networks. Since repetitiveness is the sole CCF affecting the **novelty** dimension, it also suggests a lower novelty with the UNC, for this person.

The person pointed out when answering *cognitive requirements by action*, that this

Complexity dimension	CCF	Score	Complexity relationship
Ambiguity	Input Clarity	0	Reduction
	Unstructured Guidance	1	Increase
	Process Clarity	0	Reduction
Unreliability	Inaccuracy	-1	Reduction
Incongruity	Mismatch	2	Reduction
	Heterogeneity	-1	Increase
Novelty	Repetitiveness	2	Reduction
Action complexity	Cognitive requirements by action	0	Increase

Table 5.11: Values for each complexity contributing factor, along with their associated Complexity Dimension, gathered from the answers of Subject 229. Complexity relationship specifies the relation to complexity the question the survey respondents answered.

Name	NMS Count	UNC Count	Reduction from NMS to UNC
action	116	32	72.41%
contextSwitch	60	10	83.33%
paramCount	41	16	60.98%
paramUseCount	47	20	57.45%
paramCrossContext	2	0	100%
paramAdaptCount	1	2	-100%
paramSourceScore	0	0	N/A

Table 5.12: The gathered counts for observation of 229, used for metric M2.

was the first time using the UNC, meaning it required more attention than it will do in the future. Because of this the cognitive requirement was higher using the UNC, but the person noted that in general it will be at a similar level as the current tool. This also meant the **action complexity** dimension remains similarly complex between both tools for this person.

5.5.2.2 Complexity framework

This person experienced the largest relative decrease in total actions performed, primarily due to performing 83.33% fewer context switches while using the UNC. They however modified the parameters more times when using the UNC, assigning a parameter values 1.15 times on average with the current tool, and 1.31 times with the UNC.

The difference in the amount of times parameters were modified was caused by similarly named parameters, with the user modifying a parameter, before restoring its value and modifying the correct parameter afterwards.

5.5.3 Subject 411

This person hae moderate experience using the current tool, but had not used it the last months.

5.5.3.1 Perceived Complexity

Looking at the comments noted by the researchers from the answering of the survey, this person answered positively towards the UNC on all complexity contributing factors except Heterogeneity, however, when analysing the survey results, this person answered positively towards the current tool on several of the questions. The questions answered positively about the current tool corresponds with negative statements presented in the questions, which indicates the person missed the negations. The answers to the survey can be found in (Table 5.13), and further discussed in the following paragraphs of this section.

For *input clarity*, this person perceived information given by the UNC to be more clear than information given by the current tool. The person commented that the reason for the UNC being more clear is data being structured better, leading to fewer clicks to find relevant information.

Regarding *unstructured guidance*, the person answered that guidance using the UNC was more unstructured compared to the current tool. For *process clarity*, the person answered that the UNC is more clear on how to achieve a given configuration.

Together the unstructured guidance, input, and process clarity, suggests a tendency towards the new tool having lower complexity in the **ambiguity** dimension, since both process and input clarity suggests a lower complexity in the UNC, and unstructured guidance suggests a higher complexity.

Regarding *inaccuracy* in the instructions given during configuration, this person perceived instructions to the UNC to be better, commenting that even though it was their first time, configuring using the UNC was quicker than configuring using the current tool. Since inaccuracy is the only factor affecting the **unreliability** dimension, the unreliability complexity is also lower for the UNC, for this person.

For *mismatch*, the person answered that the information presented in the UNC was easier to understand compared to the information presented in the current tool.

For *heterogeneity*, the person stated that the UNC needed more information about objects. Both through comments and through the survey response, it was clear the person viewed both tools as presenting information inconsistently.

The combination of heterogeneity suggesting no difference, and lower mismatch in the UNC, suggest the dimension **incongruity** to have lower complexity for the UNC, but still high for both tools.

Similarly, the person answered for *repetitiveness*, and therefore also the **novelty**

Complexity dimension	CCF	Score	Complexity relationship
Ambiguity	Input Clarity	3	Reduction
	Unstructured Guidance	3	Increase
	Process Clarity	2	Reduction
Unreliability	Inaccuracy	2	Reduction
Incongruity	Mismatch	3	Reduction
	Heterogeneity	0	Increase
Novelty	Repetitiveness	3	Reduction
Action complexity	Cognitive requirements by action	1	Increase

Table 5.13: Values for each complexity contributing factor, along with their associated Complexity Dimension, gathered from the answers of Subject 411. Complexity relationship reduction specifies the relation to complexity the question the survey respondents answered.

dimension, that there were fewer repetitive actions using the UNC, commenting that the existing tool was "extremely repetitive". This together with the survey response suggests a much lower complexity in the UNC.

Regarding the *cognitive requirements by action*, the person commented that both tools required substantial attention, and their survey response shows the UNC as somewhat better fitting the description of being cognitively demanding. This also suggests the person perceived a slightly higher complexity in the entire **action complexity** dimension for the UNC.

5.5.3.2 Complexity framework

Looking at the parameters collected for this person, found in (Table 5.14), all counts were lower using the UNC compared to the current tool.

When using the UNC, this user performed 80% fewer context switches compared to when using the current tool.

The total amount of variables used is 39.13% lower, and the times a parameter has been modified has been reduced even more, by 55.00%. For the current tool the average uses of each parameter was 1.7 times, and for the UNC, it is 1.3 times, meaning the person used fewer variables, and used each variable 0.4 times less.

Name	NMS Count	UNC Count	Reduction from NMS to UNC
action	140	41	70.71%
contextSwitch	80	16	80%
paramCount	23	14	39.13%
paramUseCount	40	18	55%
paramCrossContext	2	1	50%
paramAdaptCount	0	0	N/A
paramSourceScore	0	0	N/A

Table 5.14: The gathered counts for observation of 411, used for metric M2.

5.5.4 Subject 625

This person had a significant amount of experience using the current tool.

5.5.4.1 Perceived Complexity

Overall this person perceived the UNC to be less complex than the current tool, but noting neither tools were straight forward to use. While the UNC was less repetitive than the current tool, aspects of the tool design and lack of documentation regarding it meant the UNC still was difficult to use.

Another comment made by this person related to the presentation of errors given by both tools, while the UNC reported basic issues, such as submitting a string instead of a number to a parameter, more context-specific issues, such as giving several devices the same IP-address was not reported to the user until the configuration had been submitted to the NMS-API, and in some cases not at all. The person noted that using the current tool, these issues were reported to the user directly, forcing the user to always apply a technically correct configuration.

The person also noted that the overview given of changes to be made was helpful in understanding the configuration.

In the survey results found in (Table 5.15), it can be seen that the person generally regarded the UNC as being less complex than the current tool. The person explained that the biggest benefit of using the UNC over the current tool was the reduction in repetitive tasks, since it allowed for application of the same configuration on multiple devices.

For the **ambiguity** dimension, the person answered that the *input clarity*, and *process clarity* had improved with the UNC, while the UNC also increased the *unstructured guidance*. Overall this decrease in complexity because of increased clarity, combined with increased complexity because of lack of guidance lead to lower complexity in the ambiguity dimension, as perceived by this person.

This person perceived the *inaccuracy* to be similar for both tools. One reason for this inaccuracy in the UNC noted was the UNC being unclear what format some values should be submitted in, such as if a certain parameter should be written in

MHz or kHz, or if a string contained a space or an underscore. Overall this resulted in the **unreliability** dimension having similar complexity for both tools.

For the complexity dimension **incongruity**, the CCF *mismatch* reduced the complexity of UNC compared to the current tool, while the CCF *heterogeneity* showed similar complexity in both tools. Combined, this showed a lower perceived **incongruity** complexity for the UNC.

Regarding the dimension **novelty**, this person perceived the UNC better for the *repetitiveness* CCF, resulting in a slightly lower complexity for this dimension.

For *cognitive requirements by action* the person answered that both the current tool and the UNC put similar cognitive requirements on the user. The user acknowledged that the cognitive effort asserted on them was lower when using the current tool, but attributed this to their experience using the current tool. This suggests a similar complexity in the **action complexity** dimension for both tools.

Complexity dimension	CCF	Score	Complexity relationship
Ambiguity	Input Clarity	1	Reduction
	Unstructured Guidance	1	Increase
	Process Clarity	1	Reduction
Unreliability	Inaccuracy	0	Reduction
Incongruity	Mismatch	2	Reduction
	Heterogeneity	0	Increase
Novelty	Repetitiveness	1	Reduction
Action complexity	Cognitive requirements by action	0	Increase

Table 5.15: Values for each complexity contributing factor, along with their associated Complexity Dimension, gathered from the answers of Subject 625. Complexity relationship specifies the relation to complexity the question the survey respondents answered.

5.5.4.2 Complexity framework

The observations of this person working were the only one where configuring using the current tool was done with fewer total actions than using the UNC. Similar patterns could be observed when comparing paramCount and paramUseCount as well, this can be seen in (Table 5.16).

When performing the task using the UNC this person performed 290% more context switches compared to when using the current tool, this was partly due to the person performing certain tasks as part of the observation using the UNC, but not while using the current tool. However, this person also performed fewer mistakes when using the current tool compared to the UNC.

The increase in parameters used was partly due to the extra tasks performed during

Name	NMS Count	UNC Count	Reduction from NMS to UNC
action	88	211	-139.77%
contextSwitch	32	125	-290.62%
paramCount	37	47	-27.03%
paramUseCount	44	58	-31.82%
paramCrossContext	4	5	-25%
paramAdaptCount	1	0	100%
paramSourceScore	0	0	N/A

Table 5.16: The gathered counts for observation of 625, used for metric M2.

the observation of the UNC, counting only parameters part of the configuration, however accounting for this the user still modified more parameters using the UNC, compared to the current tool.

When assigning parameters values using the UNC, this person was unsure of what format several of the parameters should be submitted using, for example if certain radio frequencies should be submitted in MHz or kHz. this uncertainty meant the average amount of times a parameter was modified also increased when using the UNC instead of the current tool, being 1.19 times with the current tool and 1.23 with the UNC.

5.5.5 Subject 697

This engineer was somewhat new to the system and the current tool.

5.5.5.1 Perceived Complexity

Overall this person perceived the UNC as slightly less complex than the current tool, with a combined favourability of 0.5, on a scale from 3 to -3, with 3 being the UNC is much less complex, and -3 being the current tool is much less complex. Following, the results from the survey answered by this person, as seen in (Table 5.17) is presented, alongside comments given by them, supporting their choice of answer.

Regarding *input clarity*, this person commented it was clear in what form information should be provided to the application, thanks to the usage of drop-down menus and type hints, using the current tool; while using the UNC it was unclear how some strings should be formatted, but easier to understand which fields needed to be used.

For *unstructured guidance*, The guidance from the tools was more unstructured for the current tool compared to the UNC. The person commented that using the UNC a better overview of what information was to be filled in was given, while when using the current tool, the information was spread across multiple tabs and the person had to search through multiple tabs to find where to apply a change.

Regarding *process clarity*, the person commented that they were somewhat new to

both tools, and found both tools a bit unclear to use.

Combined, the similar complexity in both process and input clarity, along with a more unstructured guidance suggests the current tool had more complexity in the **ambiguity** dimension, compared to the UNC.

For *inaccuracy*, this person also answered that instructions given were more accurate using the current tool, no comment was given to why this answer was given, but a possible explanation is the documentation and maturity of the current tool, with company-internal documentation being available to users.

This lower inaccuracy in the current tool also suggests an overall lower complexity in the entire **unreliability** dimension, for this person.

For *mismatch*, understanding the presented information was somewhat easier using the UNC compared to the current tool.

Regarding the *heterogeneous* presentation of information in the tools, the person responded that both tools presented information equally inconsistent, and did not comment further on the subject.

The dimension **incongruity** was evaluated to be slightly less complex for the UNC for this person, since the information was easier to parse for this person, compared to using the current tool.

For *repetitiveness*, the person commented that the UNC was much less repetitive than the current tool, which is also seen in their answer on the CCF repetitiveness. Since repetitiveness is the sole contributing factor to the **novelty** dimension, it is also suggested to be less complex for the UNC

For *cognitive requirements by action*, using the current tool required more attention than the UNC according to this person, but both tools required significant attention. No comment was given to why the current tool requires more attention, or to the relative difference between the two tools, by this person. Similar to novelty and unreliability, this is the sole CCF measured for the **action complexity** dimension, suggesting a slightly higher complexity for the current tool.

5.5.5.2 Complexity framework

When this person performed the actions, they used 52.72% fewer actions with the UNC compared to the current tool, as can be seen in (Table 5.18).

The person performed 56.88% fewer context switches when using the new tool, with 67% of all actions being context switches when using the current tool, and 61% of the actions being context switches using the UNC.

Further it can be seen that the amount of parameters used was reduced by 24.32% and the parameter use count was reduced by 24.07%, meaning the average amount

Complexity dimension	CCF	Score	Complexity relationship
Ambiguity	Input Clarity	0	Reduction
	Unstructured Guidance	-1	Increase
	Process Clarity	0	Reduction
Unreliability	Inaccuracy	-1	Reduction
Incongruity	Mismatch	1	Reduction
	Heterogeneity	0	Increase
Novelty	Repetitiveness	3	Reduction
Action complexity	Cognitive requirements by action	-1	Increase

Table 5.17: Values for each complexity contributing factor, along with their associated Complexity Dimension, gathered from the answers of Subject 697. Complexity relationship specifies the relation to complexity the question the survey respondents answered.

Name	NMS Count	UNC Count	Reduction from NMS to UNC
action	239	113	52.72%
contextSwitch	160	69	56.88%
paramCount	37	28	24.32%
paramUseCount	54	41	24.07%
paramCrossContext	2	2	0%
paramAdaptCount	2	2	0%
paramSourceScore	0	0	N/A

Table 5.18: The gathered counts for observation of 697, used for metric M2.

of times a parameter was used was the same with both tools, around 1.46 times on average.

5.6 Learnings

The goal of this study was to explore methods of reducing complexity and time taken to perform configurations. During the action research cycle, a script was created to allow for easier application of complete configurations, and reuse of whole, or partial configurations, and evaluated how this new tool affected complexity and time, by observing and surveying engineers working in the current system.

5.6.1 Automation effect on time taken (RQ1)

One of the main goals of the study was to explore methods of lowering the time spent on performing and applying a configuration. The script automation tool designed was not faster than the current tool, with an average increase of 112.29%. Since the average configuration time with the current tool is 22:41, the time taken to learn and understand a new tool explains a large part of the increased time.

Also suggesting this trend is the performance of person 229, the observed person with the most previous experience with similar tools. While the average and median time increases was 10:09 and 4:56 minutes respectively, this person spent 1:54 minutes more to configure using the UNC compared to the current tool.

5.6.2 Automation effect on complexity (RQ2)

Generally, as presented in (Table 5.7), the results indicate that the respondents experienced a decrease in overall complexity. This is further supported by the overall lower values of the measured complexity parameters in (Table 5.9). Some aspects of complexity saw an increase, specifically *unstructured guidance* and *inaccuracy*, both in the "input" category.

The "input" category saw mixed results. Whilst the configuration achieved a better representation more aligned with that of the subjects, it did have a tendency to overload them unless trimmed. A benefit of UNC was the flexibility on templates to automate common configuration tasks and these were not leveraged enough in preparation for the task. It did however indicate how complex the tool was by itself without as much assisting resources, giving a slight indication on how it would be to use as a new tool when starting to implement it into the company process. The NMS also had somewhat of a similar situation during the task performance as the configuration was performed in a very empty testing environment.

UNC utilised a terminal interface and a configuration file instead of a GUI like the NMS which could be a cause to the results in the **input** dimension as explained by the work of Wong et al.[30] where they compared configurating firewall with a terminal interface and a GUI. They did not measure complexity specifically in that paper but concluded that a GUI is easier in general and even more so for beginners, while a CLI is more flexible and allows advanced users more control.

Usually it is possible to view other configurations for reference or copy parts of it when configuring a network. The evaluation task was set in a relatively empty testing environment, unfamiliar to the subjects, restricting the possibility to reference other configurations to the same extent.

A problem with the choice of interface for UNC is the increased dependency on the person performing the configuration to know and ensure that data is formatted correctly for more complex data types and following constraints that rely on other configurations not covered by UNC. As was commented on by one of the subjects, the issue arises first when you submit the configuration and not at the time of input.

It is believed that the exclusion of the memory related parameters of the complexity framework poses a non-significant risk to the conclusion of **RQ2**. The memory related aspect, albeit not observed objectively, is covered by the CCFs of the complexity model and thus has a subjective measure from the respondents. The exclusion reduces the comparability of memory load between subjects as an objective measure is more comparable than the non-linear ordinal scale of the likert scale.

A possible improvement for future iterations to further reduce complexity, is to integrate it further with the API and the editing environment. There are two different approaches possible:

- An integrated language server providing auto complete suggestions and live validation. Could be introduced in a editor as an extension or a file watch mode to better support the configuration work.

- A configuration editor with a GUI to better guide the user through the setup. Features of the NMS that assist in managing complexity could be imported and adapted to gain the benefits of both. For example showing the live network status for reference and form validation.

Process complexity appears to have improved for all subjects with one CCF remaining about the same, *cognitive requirements by action*. The lack of improvement could be explained by the lack of experience the subjects has with the tool, requiring them to focus more to learn and use it. Investigating the change of time for this CCF would yield a better understanding. It is also possible that a better abstraction for the configuration flow is achievable, taking more mental load of the one configuring the network. From the survey responses it appears a similarly cognitivel demanding but more action efficient abstraction was created.

There may also be a relations between different CCFs. That is, if other CCFs are improved, *cognitive requirements by action* may also be improved by effect. Investigating further the relationship between the identified CCFs from Peng and Li [13] could help in identifying key CCFs to target for complexity reduction. For instance, *unstructured guidance* and *inaccuracy* may have an effect on *cognitive requirements by action*, meaning that working on reducing the complexity in regards to these CCFs leads to a reduction in *cognitive requirements by action*.

5.6.3 Recommendations to Companies

Focusing on correctness, we recommend companies to explore options integrating a GUI or a language server to make configurations more aware of the environment and the more complex data types. Past research performed by Wong et al.[30] indicates that whilst advanced users may prefer terminal, a GUI is generally easier to use and

Applying system configurations in batches helps lower the complexity instead of forcing applications of configuration options one at a time. This however made it harder to understand what configuration option was incorrect when issues hindered the application of configurations.

A template creation tool could be beneficial to extract current configuration data to ease updating relevant configurations, see current configuration values and template creation.

6

Validity Evaluation

As with all research, there exist threats to the validity of the results. This chapter aims to explain what the major threats to the validity of results have been in this study, and, when applicable, what has been done to understand or mitigate the threats.

6.1 Construct validity

Every choice made as part of a scientific study risks introducing threats to the validity. Threats the researchers are aware of, that might impact the construct validity of the study, can be found in (Table 6.1).

Threat	Description	Mitigation
Non-generalizable scenario	The action taking includes the participants performing a network configuration, given a previous scenario. Since each configuration is different, there exists a possibility that the measured scenario is non-representative.	We limit the study to focus on the creation of networks using an NMS.
Negative impact on non-measured parameters	The measurement of the success of an action only takes time and complexity into account. What effect the action has on other aspects of configuration quality is not known. Examples of other measures that could be affected are: the number of errors present in a configuration, and the maintenance of tools used for configuration.	The selected parameters were chosen since they were important factors for the organisation where this study was performed. The effect the tool had on measures such as introduced errors is left for future studies to explore.
Observation related nervousness	When observing engineers, it is reasonable to assume the engineer will behave differently. The engineer might become stressed, or show themselves in a better light by working harder or being more structured than usual when under observation [31].	The researcher took part in team meetings and observed other parts of the work during diagnosing and action planning. This aimed to accustom the observed engineers to the researchers' presence before observation of the tool use. To further mitigate the effect of our conspicuous observation, both the original method and the method with an action introduced were observed in the same way.

Table 6.1

6.2 Internal validity

Threats the researchers discovered that affect the internal validity of the study, as well as how they were mitigated, can be found in (Table 6.2).

6.2.1 Issues with the tool

During the first observation as part of the action taking, bugs in the code were discovered, hindering the subject from completing one part of the task. These bugs were solved before the other observations were performed, but one observation was still affected.

The testing environment used during action planning contained old snapshots of the production environment, and the script was designed with the impression that while the current environment will be different, the types of values will remain the same. This, however, proved to be a false assumption.

Between action planning and action taking, the company moved offices, and during this move, the testing environment got updated to a newer snapshot of the production environment. The researchers performed some tests to ensure the action was still possible to perform, but this testing did not cover every possible scenario.

During the first observation as part of action taking, it was noticed that certain fields containing lists of configurations could either be populated with values, contain an empty list, or be missing entirely from the configuration. Previously, all fields had a value assigned, even if that value was an empty list or a string of length 0. In some cases, these inexistent fields would cause the script to fail.

Since the issue arose when applying a new configuration to certain devices, the researchers decided to continue the observation, pretending the new configuration was applied correctly, and a patch was applied to solve the issue before the following observations took place.

The discovered threat caused by the tool can be found in (Table 6.3)

6.2.2 Likert scales

Likert scales are subject to several different kinds of bias. In table (Table 6.4) we will address these biases and how we accounted for them.

Threat	Description	Mitigation
Organisational move	Between the action planning and action taking phases, the organisation moved offices, and as part of this, the network configuration changed. This change in physical and computational environment may have had an effect on the part of the configuration that was analysed, and the action was planned to improve.	To understand the effects of the move, the researchers were in contact with the reference team and discussed the changes.
Changes to the system	Between taking measurements of the UNC, and the current tool, the state of the system was partially restored, but some artefacts modified were not correctly restored after each measurement was taken. This resulted in participants reusing configuration artefacts created when measuring UNC during the measurement of the current tool.	The participants were told not to reuse artefacts created during the previous observation. If a participant still reused an artefact, it was noted and presented in the results section of the thesis.
Miscalculation of parameters	The collection of data for Brown's complexity framework [7], as well as for measuring the time a configuration takes, was collected manually. Since the collection process takes more than one hour per participant and tool, each metric was only collected from the videos once. There exists a possibility that some events have been missed or counted twice.	For parameters where the difference between results was small (less than 5 actions), the conclusivity of the result was questioned by the researchers and deemed to be similar in complexity.
Second think	By observing the engineers work, and by asking them questions about the work they do, primarily as part of diagnosing, the engineers might question the way of working, and change their workflow at the same time as the action taking is performed. This can lead to a change in the performance of engineers, not because of the action, but still measured during action taking.	To mitigate the effects of this, the researchers decided to perform the observations directly after each other, not providing sufficient time between to analyse what was done as part of the observation.

Table 6.2

Threat	Description	Mitigation
Tool update	Between the first observation and the three later observations, the UNC was modified to fix a bug in the code.	When the subject encountered the bug, they were told to continue as if the issue had not occurred. Since the step interrupted by the issue was applying the configuration to some devices, the workflow still remained the same during this observation, with only a difference in resulted configuration.

Table 6.3

Threat	Description	Mitigation
Censoring of beliefs	The limiting of respondents' answers to a limited set of answers lowers the precision of said answers. This becomes especially apparent towards the extremes [32].	Elicitation of Likert-type responses was accompanied by an observation where the respondents were encouraged to describe their reasoning behind selecting their response.
Central tendency bias	Giving the option of selecting a neutral, or no opinion- option makes it difficult to interpret the responses to a survey [33]. Whether a person selected the neutral option to avoid answering a question or if they truly are neutral is hard to know.	Surveys still contained an odd number of choices for respondents to select between, to avoid forcing an opinion where none existed, but more degrees of answers were given than is standard, to encourage answering either direction with less leaning than normal.
Choosing between extremes	With an even number of answer choices, respondents are forced to answer positively or negatively on a given question [33]. For more complex questions, the respondent might not have an opinion but is still forced to answer with one.	By accompanying the survey with an observation, it was possible for respondents to orally mention their lack of opinion and include an odd number of answer options.
Social pressure	If the distributor of the survey influences the respondent, it is likely some respondents will give answers not based on their beliefs, but rather their beliefs of what society, or the survey distributor, wants as a result [33].	The researchers ensured respondents' anonymity of answers towards persons who have influence over them.

Table 6.4

6.3 External validity

Threats the researchers discovered that affect the external validity of the study, as well as how they were mitigated, can be found in (Table 6.5).

Threat	Description	Mitigation
Specific context	This study was performed within a specific context, and with a small sample size, it's a possibility that applying the same learnings within a company with a different culture or workflow, or with engineers with different previous knowledge, the results will be different.	The researchers avoid claiming generalisations of the study on contexts outside of the specific company. While it is possible similar results will appear in other organisations, more studies are required to achieve greater generalisations.
Test-specific result	Since the tool was compared on a not disclosed network of devices, and against an existing tool, the results might not represent what is perceived with device networks of different structures, and when compared against different existing tools.	The researchers avoid making generalisations across all network sizes and tools. This specific study covers the effects of automating parts of the configuration process on a smaller network.

Table 6.5

7

Conclusion

This study has examined the impact of script automation on software configuration within a medium-sized company, measuring the script automation's effect on configuration complexity and time. A tool was developed and evaluated.

For the subjects that participated in the study, the survey indicate a lower overall complexity with the tool, though some subjects experienced an increase in complexity in some dimensions. One subject experienced an increase in measured complexity.

To answer the first research question *What effect does automation have on the time it takes to configure a new network?* Within the context of the medium-sized SaaS company, automating parts of the configuration of networks indicate an increase in time taken to perform the configuration for all subjects. Partially this increase is believed to be caused by participants having previous experience with the current tool, but not with the UNC. However, there exists no evidence for the UNC to be faster, or similarly fast, to use. A longer investigation would need to be performed to conclude whether UNC is capable of providing a decrease in configuration time.

To answer the second research question *How does the perceived complexity of configuring a network change when using automation?* In general, the subjects experienced a decrease in complexity, both as perceived, and as measured. There was however one subject who experienced an increased measured complexity. It was also noticed that the results suggest that the overall perceived complexity decreased, certain dimensions of complexity increased, namely the **unstructured guidance** and the **inaccuracy** dimensions.

7.1 Future work

Due to time constraints, the study evaluated the tool at one point in time to better understand the effects of the tool. Further observations should be performed to mitigate the effects of differing experiences between different tools.

The study focused on how configuration complexity and configuration time were affected by using script automation; there are, however, other aspects which effects the feasibility of using script automation for configuration, such as the impact on configuration errors and how the maintainability of the system changes.

The results from the first cycle suggest the UNC has the ability to lower the complexity of the task, the time a configuration takes has not improved using the UNC. As can be seen in observations of subjects 229 and 411, much of the time difference comes from a lack of experience using tools similar to the UNC. A future cycle will benefit from focusing on documentation and improving the user experience by providing better templates depending on the task at hand and structuring the workflow further.

These points were not able to be explored further in a second cycle as the research was limited to a 6 month period and action research cycles take 3-6 months. A second cycle could expand the dataset to allow more certainty in the conclusion and improve the documentation and support for new users to the tool.

Bibliography

[1] Fernando Caicedo-Altamirano et al. "Network Automation Using Python Scripts with the Telnetlib Library and the Telnet Protocol". In: *International Conference on Computer Science, Electronics and Industrial Engineering (CSEI)*. Springer. 2024, pp. 777–788.

[2] Hongqiang Harry Liu et al. "Automatic life cycle management of network configurations". In: *Proceedings of the Afternoon Workshop on Self-Driving Networks*. 2018, pp. 29–35.

[3] Tianyin Xu and Yuanyuan Zhou. "Systems approaches to tackling configuration errors: A survey". In: *ACM Computing Surveys (CSUR)* 47.4 (2015), pp. 1–41.

[4] Mohammed Sayagh et al. "Software Configuration Engineering in Practice Interviews, Survey, and Systematic Literature Review". In: *IEEE Transactions on Software Engineering* 46.6 (2020), pp. 646–673. DOI: 10.1109/TSE.2018.2867847.

[5] Zuoning Yin et al. "An empirical study on configuration errors in commercial and open source systems". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: Association for Computing Machinery, 2011, pp. 159–172. ISBN: 9781450309776. DOI: 10.1145/2043556.2043572. URL: https://doi.org/10.1145/2043556.2043572.

[6] David Plonka and Andres Jaan Tack. "An Analysis of Network Configuration Artifacts." In: *LISA*. 2009, pp. 79–91.

[7] Aaron B Brown, Alexander Keller, and Joseph L Hellerstein. "A model of configuration complexity and its application to a change management system". In: *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.* IEEE. 2005, pp. 631–644.

[8] Steve McConnell. *Code complete*. Pearson Education, 2004.

[9] en. In: *Oxford English Dictionary*. 3rd ed. Oxford University Press, Mar. 2023. DOI: 10.1093/OED/9697864014. URL: https://oed.com/dictionary/complexity_n.

[10] Ms Priti V Wadal and SR Gupta. "An Overview of Network Management System". In: *International Journal Of Computer Science And Applications* 6.2 (2013), pp. 0974–1011.

[11] Markus Sebeck and Gerd Bumiller. "A network management system for powerline communications and its verification by simulation". In: *4th International Symposium on Power Line Communications and Its Applications*. 2000, pp. 225–232.

[12] OpenNMS Group. *Administrators Guide*. (Accessed 26-03-2026). May 2016. URL: https://vault.opennms.com/docs/opennms/releases/18.0.0/guide-admin/guide-admin.html.

[13] Peng Liu and Zhizhong Li. "Task complexity: A review and conceptualization framework". In: *International Journal of Industrial Ergonomics* 42.6 (2012), pp. 553–568. ISSN: 0169-8141. DOI: https://doi.org/10.1016/j.ergon.2012.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0169814112000868.

[14] Peng Liu and Zhizhong Li. "Comparison between conventional and digital nuclear power plant main control rooms: A task complexity perspective, Part II: Detailed results and analysis". In: *International Journal of Industrial Ergonomics* 51 (2016). Human Factors in Digital Industrial Systems, pp. 10–20. ISSN: 0169-8141. DOI: https://doi.org/10.1016/j.ergon.2014.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0169814114001085.

[15] Felipe A Lopes et al. "A software engineering perspective on SDN programmability". In: *IEEE communications surveys & tutorials* 18.2 (2015), pp. 1255–1272.

[16] Wenfeng Xia et al. "A survey on software-defined networking". In: *IEEE Communications Surveys & Tutorials* 17.1 (2014), pp. 27–51.

[17] Mahmoud Abbasi, Javier Prieto, and Juan Manuel Corchado. "Network automation: from intent-based networking to cloud-native networking". In: *International Symposium on Distributed Computing and Artificial Intelligence*. Springer. 2023, pp. 418–427.

[18] Marina Gutiérrez et al. "Self-configuration of IEEE 802.1 TSN networks". In: *2017 22nd IEEE international conference on emerging technologies and factory automation (ETFA)*. IEEE. 2017, pp. 1–8.

[19] Artem Voronkov, Leonardo A Martucci, and Stefan Lindskog. "System administrators prefer command line interfaces, don't they? an exploratory study of firewall interfaces". In: *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 2019, pp. 259–271.

[20] Wei Wei. "The impact of product complexity on adoption of web-based interactive innovation practices". In: *Innovation* 14.3 (2012), pp. 431–445.

[21] Fred D Davis et al. "Technology acceptance model: TAM". In: *Al-Suqri, MN, Al-Aufi, AS: Information seeking behavior and technology adoption* 205.219 (1989), p. 5.

[22] Sihyung Lee, Tina Wong, and Hyong S Kim. “To automate or not to automate: on the complexity of network configuration”. In: *2008 IEEE international conference on communications*. IEEE. 2008, pp. 5726–5731.

[23] Aaron B Brown and Joseph L Hellerstein. “Reducing the cost of it operations-is automation always the answer?” In: *HotOS*. 2005.

[24] Miroslaw Staron. *Action research in software engineering: Theory and applications*. Springer, 2020.

[25] Sirwan Khalid Ahmed et al. “Using thematic analysis in qualitative research”. In: *Journal of Medicine, Surgery, and Public Health* 6 (2025), p. 100198. ISSN: 2949-916X. DOI: https://doi.org/10.1016/j.glmedi.2025.100198. URL: https://www.sciencedirect.com/science/article/pii/S2949916X25000222.

[26] Rini Solingen and Egon Berghout. “The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development”. In: (Jan. 1999).

[27] Katherine A. Batterton and Kimberly N. Hale. “The Likert Scale What It Is and How To Use It”. In: *Phalanx* 50.2 (2017), pp. 32–39. ISSN: 01951920. URL: http://www.jstor.org/stable/26296382 (visited on 02/24/2026).

[28] Mark W Maier. *The art of systems architecting*. CRC press, 2009.

[29] Connie Smith and Lloyd Williams. “More New Software Antipatterns: Even More Ways to Shoot Yourself in the Foot.” In: Dec. 2003, pp. 717–725.

[30] Tina Wong. “On the usability of firewall configuration”. In: *Symposium on usable privacy and security*. 2008.

[31] Leah Brackett, Dennis H Reid, and Carolyn W Green. “Effects of reactivity to observations on staff performance”. In: *Journal of applied behavior analysis* 40.1 (2007), pp. 191–195.

[32] J Christopher Westland. “Information loss and bias in likert survey responses”. In: *PloS one* 17.7 (2022), e0271949.

[33] Imam Kusmaryono, Dyana Wijayanti, and Hevy Risqi Maharani. “Number of response options, reliability, validity, and potential bias in the use of the Likert scale education and social science research: A literature review”. In: *International Journal of Educational Methodology* 8.4 (2022), pp. 625–637.

A

Protocols

A.1 Diagnosing Interview Protocols

The questions used for the interview protocols are presented hereafter.

A.1.1 Network configuration experience

Questions about personal experience with satellite network setup
Question: *How long have you been in the systems team?*

Question: *How many times, if any, have you setup a new satellite network?*

Question: *Describe your work process when you set up new satellite networks.*

Question: *How do you keep track of what changes you need to make in the NMS to setup the new satellite network?*

Question: *When setting up a new satellite network, what do you find to be the most difficult?*

Question: *Do you use anything to help with the configuration of the NMS?*

Questions about time
Question: *How much time does it usually take to setup a new network? Both working time and total time.*

Question: *Are there any tasks that are more time consuming?*

A.2 Diagnosing Observation Protocols

The protocols used for the observations are presented hereafter with their title describing the event.

A.2.1 Setup of a new network

Item	Details
Task 1 was in from the checklist was completed	Do not forget time
Task 2 was in from the checklist was completed	Do not forget time
Task 3 was in from the checklist was completed	Do not forget time
Task 4 was in from the checklist was completed	Do not forget time
Task 5 was in from the checklist was completed	Do not forget time
Task 6 was in from the checklist was completed	Do not forget time
Task 7 was in from the checklist was completed	Do not forget time
Task 8 was in from the checklist was completed	Do not forget time
A step in the checklist was missed	
One or more steps were not completed in their entirety	
iBuilder notifies of one or more incorrect values	
One or more incorrect values was accepted by iBuilder	
One or more incorrect components gets selected	
Two or more people try using iBuilder at the same time	
The engineer is uncertain if an option is necessary	

A.2.2 Updating the configuration of an existing network

Item	Details
One or more incorrect values are accepted by iBuilder	
The engineer has trouble finding the correct fields to update	
iBuilder warns about incorrect values	

B

Detailed implementation descriptions

B.1 Validation module

One use case of the **validation** module is to validate the correctness of an IP address as can be seen in (Figure B.1), where a field containing an IP address is provided, and a boolean value is returned, showing whether the field contains a valid IP or not. Since the system uses the IPv4 addressing system, the field is split into 4 sections, divided by dots. If the field was not split into 4 different parts when splitting on dots, or one or more of the parts contains values outside of the accepted range of $[0, \ldots, 255]$, false is returned, otherwise the function returns true.

```
1 class ValidationHelper:
2     @classmethod
3     def is_ip_address(cls, value: str) -> bool:
4         parts = value.split(".")
5         return len(parts) == 4 and all(part.isdigit() and 0 <= int(
    part) <= 255 for part in parts)
6
```

Figure B.1: Method validating if a string is formatted as an IP address. Returns True if the provided field is a valid IP address, otherwise returns False.

B.2 Client modules

The `DryRunClient`, seen in (Figure B.2) analyses all requests sent through the client, routing all GET-requests to the provided host, but intercepting all other requests, using the logger mentioned previously to present the user with information about the request structure and information, without routing it to the host.

Both the `HttpsClient` and the `DryRunClient` can be provided a hostname to allow for switching between being used in a testing environment and in a real-world environment, where the testing environment can be used to either test new versions of the script, test a configuration, or to test the functionality of other softwares in networks configured in different ways.

```python
class DryRunClient(HttpsClient):
    def _send(
        self,
        method: str,
        path: str,
        headers: dict[str, str] = {},
        body: str | None = None,
    ) -> "Response":
        if method == "GET":
            return super()._send(method, path, headers=headers, body=body)
        else:
            Printer.info(
                f"Dry run client is used - request would be sent to {self.hostname}:{self.port}{self.base_path}{path} with method {method} and body:\n{body}"
            )
            return Response({}, 200, "OK")
```

Figure B.2: The dry-run client extension to the https-client, used to visualize requests sent to an API without applying those changes.