Tampereen yliopisto

Iiris Lilja

USING AND ADVANCING DITA XML

A case study of evaluating DITA-authored customer documentation

Faculty of Information Technology and Communication Sciences
Master's Thesis
April 2022

ABSTRACT

LILJA, IIRIS: Using and advancing DITA XML – A case study of evaluating DITA-authored customer documentation
Master's thesis
Tampere University
Master's Programme in Multilingual Communication and Translation Studies (English translation and interpreting)
April 2022

In today's world, customer documentation for products must comply with the changing needs of the users, who want to access and consume information effortlessly as soon as they need it. Traditional documentation strategies such as desktop publishing and linear writing have been challenged by the emerging technologies in technical communication that emphasize modularity, flexibility and efficiency from both the users and the technical writers' perspectives. The Darwin Information Typing Architecture (DITA) is one of the most interesting technologies within the technical communication field.

The theoretical framework of this study is based on markup languages – especially DITA – and various perspectives of technical writing, such as topic-based writing, single-sourcing and minimalism. The features of DITA and the best practices of structured writing offer a lucrative starting point for addressing the highly practical research question of this study, namely how DITA XML has been used. This thesis is done in collaboration with a large company's business unit that produces highly complex products and whose technical documentation strategy includes writing use case-based DITA topics. Now, the unit has initiated a transformation project whose goal is to evolve the customer documentation, and this thesis is a part of this project.

The research problem of this thesis was approached by conducting a heuristic evaluation by using the DITA heuristics that were created as a part of this study, as such heuristics did not exist yet. The heuristics are based on both literature and practical experience on authoring technical documentation in DITA, as well as the business unit's internal guidelines and discussions with a senior documentation specialist with extensive knowledge and competencies of the themes of this thesis. The DITA heuristics will later serve as a basis for a tool for the technical writers at the unit. The research data comprises 80 DITA XML modules, 40 of which contain descriptive, conceptual information (DITA concept topics) while the remaining 40 guide the users on carrying out procedures (DITA task topics). The sample data dates to 2021, and it was collected from the customer documentation of four different software components.

The research shows that DITA can be used in various ways to author technical documentation, but some practices conform better to the principles of the DITA standard as well as the guidelines of structured, modular documentation. The most central issues include semantic markup, information typing and minimalism, especially task-orientation. Moreover, conducting a heuristic evaluation with a customized heuristics list proved to be an effective method for assessing the practices of authoring customer documentation in DITA. The results of this thesis provide valuable information for the business unit's endeavors in terms of evolving their customer documentation content and presentation as well as supporting technical writers.

The results of this thesis can be used internally in the business unit as a part of their transformation project regarding the usability and user-centeredness of their customer documentation. As a continuum to this study, technical writers who use DITA could be researched in terms of how they perceive DITA and structured authoring to increase the knowledge of how these themes are regarded among technical documentation experts today.

Keywords: DITA, XML, heuristic evaluation, minimalism, modularity, technical documentation

The originality of this thesis has been checked using the Turnitin Originality Check service.

TIIVISTELMÄ

LILJA, IIRIS: DITA XML:n käyttö ja kehitys – tapaustutkimus asiakasdokumentaatiosta
Pro gradu -tutkielma
Tampereen yliopisto
Monikielisen viestinnän ja käännöstieteen maisteriohjelma, englannin kääntämisen ja tulkkauksen opintosuunta
Huhtikuu 2022

Asiakasdokumentaation käyttäjät sekä heidän tapansa ja tarpeensa ovat muuttuneet ajan saatossa, minkä seurauksena myös itse dokumentaatio on muuttunut. Teknisen viestinnän alalla on havaittu, että käyttäjät haluavat saada tarvitsemansa tiedon nopeasti juuri sillä hetkellä, kun he sitä tarvitsevat. Tästä syystä perinteinen lineaarinen dokumentaatio, jossa käyttöohjeet kirjoitetaan kokonaisuutena alusta loppuun sivu-sivulta-periaatteella on alkanut väistyä uusien metodien tieltä, jotka painottavat modulaarisuutta, joustavuutta ja tehokkuutta niin käyttäjien kuin informaatiota tuottavien teknisten kirjoittajienkin näkökulmasta. Yksi alan suosituimmista tähän tarkoitukseen sopivista metodeista on Darwin Information Typing Architecture eli DITA.

Tämän pro gradu -tutkielman teoreettinen viitekehys rakentuu merkintäkielten (etenkin DITAn) ja teknisen viestinnän eri periaatteiden ja hyvien käytäntöjen ympärille, joihin sisältyvät modulaarisuus, yksilähteistäminen ja minimalismi. Tutkimuskysymys on hyvinkin käytännöllinen, nimittäin tässä tutkielmassa tarkastellaan DITA XML:llä tuotettua dokumentaatiota sen lähtöformaatissa. Tutkielma on tehty yhteistyössä erään suuren yrityksen kompleksisia tuotteita valmistavan liiketoimintayksikön kanssa, jossa olen ollut harjoittelussa teknisenä kirjoittajana. Yksikössä tuotetaan itse tuotteiden lisäksi myös niiden dokumentaatio, ja DITA on tämän dokumentointistrategian ytimessä. Yksikössä on käynnistetty transformaatioprojekti, jonka tarkoituksena on kehittää asiakasdokumentaatiota entisestään. Tämä tutkielma on osa tuota projektia.

Tutkimuskysymystä lähestyttiin toteuttamalla heuristinen arviointi dokumentaatiolle. Tutkielmassa esitellyt ja käytetyt DITA-heuristiikat luotiin osana tutkimusta, sillä vastaavia heuristiikkoja ei ollut luotu aiemmin. Heuristiikat pohjautuvat teoreettiseen ja käytännön tuntemukseen dokumentoinnista DITAa käyttäen, minkä lisäksi niiden taustalla vaikuttavat yksikön sisäiset ohjeistukset ja käytänteet sekä keskustelut ja yhteistyö yksikön dokumentaatioasiantuntijan kanssa, jolla on kattava kokemus ja tietämys tutkielmaan liittyvistä aihepiireistä. Tutkimusaineisto kattaa 80 DITA-topiikkia eli yksittäistä moduulia neljästä eri ohjelmistotuotekomponentista. Topiikeista 40 käsittelee konseptuaalista informaatiota komponenteista, kun taas toiset 40 kuvailevat tehtäviä, joita komponenttien käyttäjät noudattavat. Aineisto on peräisin vuodelta 2021.

Tutkimuksen tulokset osoittavat, että DITAa voidaan soveltaa monilla eri tavoilla dokumentaatiossa, mutta tietyt tavat toteuttavat DITAn arkkitehtuuria muita paremmin. Lisäksi tapojen välillä havaittiin olevan eroja sen suhteen, miten hyvin ne vastaavat rakenteisen ja modulaarisen tekstin kirjoittamisen konventioita. Olennaisimmat havainnot liittyvät semanttisen merkintäkielen käyttöön, minimalismiin ja informaatiotyypittelyyn sekä käyttäjän ydintehtävien korostamiseen tuoteominaisuuksiin keskittymisen sijaan. Tutkimuksen johtopäätökset tarjoavat käyttökelpoista tietoa yksikön transformaatioprojektin tarpeisiin, ja niitä tullaankin käyttämään osana asiakasdokumentaation kehittämistä.

Tutkimuksen perusteella jatkotutkimuksen aiheena voisi olla esimerkiksi DITAa käyttävien teknisten kirjoittajien suhtautuminen tässä tutkielmassa käsiteltyihin rakenteellisen kirjoittamisen ja modulaarisuuden teemoihin. Näin olisi mahdollista saada ajankohtaista tietoa siitä, miten alan ammattilaiset näkevät DITAn ja sen käyttökelpoisuuden.

Avainsanat: DITA, XML, heuristinen arviointi, minimalismi, modulaarisuus, tekninen dokumentaatio

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -ohjelmalla.

TABLE OF CONTENTS

1 Introduction 1

2 DITA in technical communication 3

2.1 Markup languages 3

2.2 DITA XML 8

2.2.1 A structured authoring platform 10

2.2.2 A topic-based information architecture 12

2.2.3 A writing methodology 15

2.3 Main features of DITA – inheritance, specialization and reuse 17

2.4 DITA and minimalism 23

2.5 Benefits and challenges of DITA 27

3 Heuristic evaluation 33

3.1 Heuristic evaluation as a method 33

3.2 Heuristics for DITA 36

4 Data 45

4.1 Customer documentation in DITA 45

4.2 Sample size and collection 46

5 Heuristic evaluation of DITA topics 49

5.1 Information typing and modular writing 50

5.2 Formatting titles 54

5.3 Using elements 57

5.4 Separating content from form 61

5.5 Reusing content 63

5.6 Linking and connecting information 65

5.7 Writing minimalist and task-orientated topics 68

5.8 Summary of the evaluation 71

6 Conclusions 76

Works cited 79

Suomenkielinen lyhennelmä i

1 INTRODUCTION

Technical documentation is less and less about writing linear customer documentation with word processors such as Microsoft Word; it is increasingly more about writing structured content that is modular, reusable, and scalable to all projects and products regardless of their size. Customer documentation, also called *information* or *instructions for use*, is "the information provided by the supplier that provides the target audience with concepts, procedures and reference material for the safe, effective, and efficient use of a supported product during its life cycle" (SFS-EN IEC/IEEE 82079-1 2020, 76). Creating this type of documentation often requires dedicated methods. One such method is the Darwin Information Typing Architecture (DITA), which is a widely used semantic Extensible Markup Language (XML) architecture for producing content, especially technical documentation that conforms to a predetermined standard. But what exactly is DITA, and how does it work? DITA is a standard that defines a set of document types for authoring and organizing topic-based, information-typed, structured content that can be reused and single-sourced in various ways as well as published in multiple media formats (OASIS 2015, 20). While DITA is far from being the ubiquitous default choice for technical documentation, its appeal among a broad number of organizations cannot be dismissed (Schengili-Roberts 2014). DITA is an attractive documentation strategy for businesses thanks to the benefits of XML itself as well as the many useful abilities of DITA specifically, including reusing and repurposing content, improved efficiency and reduced localization costs (Priestley and Swope 2008, 3).

This thesis is conducted for a business unit in a large company with extensive, early adopter experience of using DITA for nearly two decades. As a result, their DITA strategy encompasses many beneficial DITA practices, such as content reuse, minimalist writing and multi-channel publishing. However, due to the large size and various mergers and acquisitions by the company, the unit's DITA strategy across its various product components has diffused internally into co-existing (and sometimes contradictory) parallel DITA practices that do not conform to the design of DITA or the unit's internal guidelines.

Now, the unit has launched an internal transformation project, and one of its goals is to enhance the end-user experience of the customer documentation by exploiting DITA among other actions. This thesis is connected to this project, and due to my technical writing traineeship at

the unit, I was intrigued by the possibility of conducting research as a part of this process. The research problem in this qualitative case study is of a practical nature: to assess the DITA-authored customer documentation of four of the many components of the business unit to form a data-based understanding of how DITA XML has been used. The results of the study will be used internally to help focus on the most urgent needs of the documentation teams.

The theoretical framework of this thesis consists of literature on technical documentation, markup languages (specifically DITA), information design and usability. The research material comprises a sample of 80 DITA XML topics of the business unit's customer documentation for four product components, which are included in the transformation project. The research problem is approached by combining theoretical and practical knowledge of DITA, minimalism and structured, modular, topic-based writing to create the DITA heuristics list specifically for the business unit's needs and using it to conduct a heuristic evaluation on the 80 sample DITA topics. Although various lists of heuristics exist, currently none are dedicated for DITA or structured writing in general, which is why I compiled the heuristics based on my prior experience of conducting a heuristic evaluation as well as my technical writing traineeship in the business unit. The DITA heuristics were created as an iterative process in collaboration with a senior documentation specialist of the business unit with extensive knowledge of the subject.

Research into DITA typically focuses on its principles and benefits or the early phases of starting to use the method. So far, there has been little practical research into advancing DITA in organizations with vast experience of using it. Moreover, practical research issues have been identified as one of the most topical questions for the technical communication field that has grown with the demand for writers who can enable people to use sophisticated technologies and specialized knowledge (Rude 2009, 198–199). Among others, the themes of developing and managing information, including structured authoring and single-sourcing are potential research issues (ibid.). These issues are also major points of interest for the business unit, and as a consequence they are reflected in the chosen research method and the customized DITA heuristics.

This thesis has six main chapters. Chapter 2 outlines markup languages, structured documentation, and DITA XML along with its features, benefits, challenges and relation to minimalism. Chapter 3 introduces the principles of heuristic evaluation and the DITA heuristics that were created for this study. The research data is presented in chapter 4, after which it is analyzed in chapter 5. Finally, the conclusions of the study are discussed in chapter 6.

2 DITA IN TECHNICAL COMMUNICATION

"Developed by technical communicators, for technical communicators" is how the Darwin Information Typing Architecture is described in the DITA Style Guide (Self 2011, 220). Technical communicators – or writers, as they are titled in this thesis – are experts who possess a variety of skills and competencies needed to define, create and deliver information products through the process of technical documentation for the safe, efficient and effective use of different types of products, such as technical systems and software (Tekom Europe 2022). In addition to the knowledge and skills, writers also need tools. Within the tools domain, the DITA standard has been and continues to be a prevalent choice for organizations who seek a way to produce XML-based modular and structured content. As Day et al. describe (2001, 1), "DITA is an XML-based, end-to-end architecture for authoring, producing and delivering technical information", and "[t]his architecture consists of a set of design principles for creating 'information-typed' modules at a topic level." This chapter is a roadmap to markup languages and DITA, which are essential subjects in this thesis. In addition, this chapter covers the core principles and features of DITA as well as the benefits and challenges associated with it.

2.1 Markup languages

Extensible Markup Language (XML) is a standard for defining markup languages, which are human- and machine-readable computer languages of start- and end-tags that complement the text in a document with information about the text components. XML is not a markup language itself; it is a language for developing markup languages. The XML specification was published by the World Wide Web Consortium in 1998 as a modernized version of Standard Generalized Markup Language (SGML, ISO8879:1986), an older and broader metalanguage for defining markup languages (Priestley et al. 2001, 352). SGML-based Hypertext Markup Language (HTML), which is used for creating and displaying web pages and applications, is perhaps the most well-known markup language. HTML was released in 1993, and since then HTML text files have been the core components of all web pages (Wikipedia, 2022b). Markup languages consist of *tags*, which are markup components used for identifying different types of data by carrying information. There are two types of tags, both of which are written within angle brackets (< >). Start-tags begin with < and end with > (e.g. <body>), while end-tags start with < and a forward slash / and end with a > (e.g. </body>). XML tags are case-sensitive, which is

why the syntax (the structure of statements) of example 1 is correct but the syntax of example 2 is wrong:

(1) <example>This is correct XML syntax.</example>
(2) <example>This is wrong XML syntax.</Example>

The syntax of example 2 is incorrect because the case in the </Example> end-tag differs from that of the <example> start-tag. The tag set and the in-between textual content is called an *element*. Elements are the building blocks of markup languages, and they can include either text (with the exception of the word *xml*) or other elements (Katajisto 2020a). All XML documents have a *root element*, which encloses all the other elements of that document. Elements can also have *attributes* (marked with the @ symbol) whose purpose is to contain more specific information about the element. For example, the @product attribute can be used to specify the product in question. Attributes can be used for determining how the content should be conditionally processed, including whether it should be filtered (excluded or included); flagged (highlighted); or passed through, which means including the content in the output and preserving the attribute for further processing (OASIS 2015, 50; 343).

XML-based markup languages differ from SGML-based ones such as HTML in the sense that they describe what the data is while HTML, for example, emphasizes what the data looks like in terms of basic design, including font size, formatting and heading levels. Another important distinction is that unlike HTML, XML does not have predefined elements, because they are created by the developer of the XML document. The X in XML stands for *Extensible*, which means that an XML-based markup language can be extended to meet the requirements of the developer. Anyone can invent their own markup language and define their own entities in a *document type definition* (DTD) file, which can be declared either as external file references or internal declarations within the XML document. The DTD is associated with XML and SGML documents via the document type declaration (DOCTYPE) that is located at the start of the document as is shown in example 3 where the DTD called “topic” is declared externally with a reference to the topic.dtd file:

(3) <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA Topic//EN" "topic.dtd">

As Priestley et al. (2001, 353) explain, the DTD defines the allowable markup and markup rules for the respective markup language, and the core of the XML standard is essentially a set of rules and guidelines for creating DTDs. The rules in the DTD define the allowed elements and attributes as well as their order and hierarchy. For example, the XHTML DTD defines the

<head> and <body> elements and that <head> cannot be inserted after <body> (Priestley et al. 2001, 353). If the first paragraph of this chapter were written in a markup language, example 4 shows one of the infinite possibilities of what it could look like:

(4)
```
<chapter>
  <title>Markup languages</title>
  <body>
    <p>Extensible Markup Language (XML) is a standard for defining markup languages, which are human- and machine-readable computer languages of start- and end-tags…</p>
  </body>
</chapter>
```

Although example 4 does not represent any existing XML, its syntax illustrates the *semantic* nature of XML. In linguistics, semantics is the study of the meaning of words and how they are interpreted. The attributes and elements of XML-based markup languages use human-readable, standard words that are applied to content based on what kind of information the content represents. This enables computers to not only process text, but also process information about the text, and that information is understandable to humans, too. Semantic markup also improves search engine results (ibid.).

The pseudo-XML of example 4 is not associated with any DTD. The importance of DTDs is clarified by Heikniemi (2001), who explains the difference between *well-formed* and *valid* XML: if an XML document has been constructed according to the XML syntax rules (for example, having complete start- and end-tags in the correct order; nesting elements according to the specifications), it is well formed. Example 4 represents well-formed XML. However, well-formedness alone does not suffice in the long term, because it does not guarantee that the structure of the XML document is free of errors (ibid.). The purpose of a DTD is to ensure that information flows logically and is presented in the appropriate order – for instance, a DTD for the above example would define the <chapter> and <title> elements and state that <title> must be nested inside <chapter> and that <title> must be the first element nested under <chapter>. An XML document is valid when it adheres to both general XML syntax and the respective DTD rules (ibid.). If the text of example 4 were published in a delivery format, it would also require a stylesheet (Cascading Stylesheet, CSS) or an Extensible Style Language (XSL) to map the XML elements to display textual properties, such as font type and size (Priestley et al. 2001, 353). Furthermore, an Extensible Stylesheet Language Transformations (XSLT) stylesheet would be needed for transforming the text into plain, unformatted text or another XML structure, which in turn allows transforming the XML source into formats such as HTML

or PDF (Priestley et al. 2001, 353). These methods are needed because XML documents separate content from form to be delivery-agnostic and free of as much context (how the information will be used, presented or sequenced) as possible (Self 2011, 125), which enables creating an infinite number of outputs from the same source file. As Priestley et al. (2001, 353) state, XML includes many promises, but the solution is not a very simple one, and it also requires a significant amount of maintaining.

To summarize, an XML document is created by using an XML-based markup language, typically with an XML editor. Technically, any text editor can edit XML, but some software programs are specialized for the task and support various useful XML authoring features. This thesis assumes that Oxygen XML Author (styled as *<oXygen/>*) developed by SyncRO Soft SRL (from hereon, oXygen) is used, as it supports DITA and is the XML authoring tool used in the business unit. The tool is a visual, cross-platform application for editing, authoring and publishing XML documents using structured markup languages, especially DITA XML. The content is displayed in a graphical user interface (GUI), which makes editing user-friendlier (Self 2011, 225). Figure 1 shows the Author mode view of XML content in oXygen:

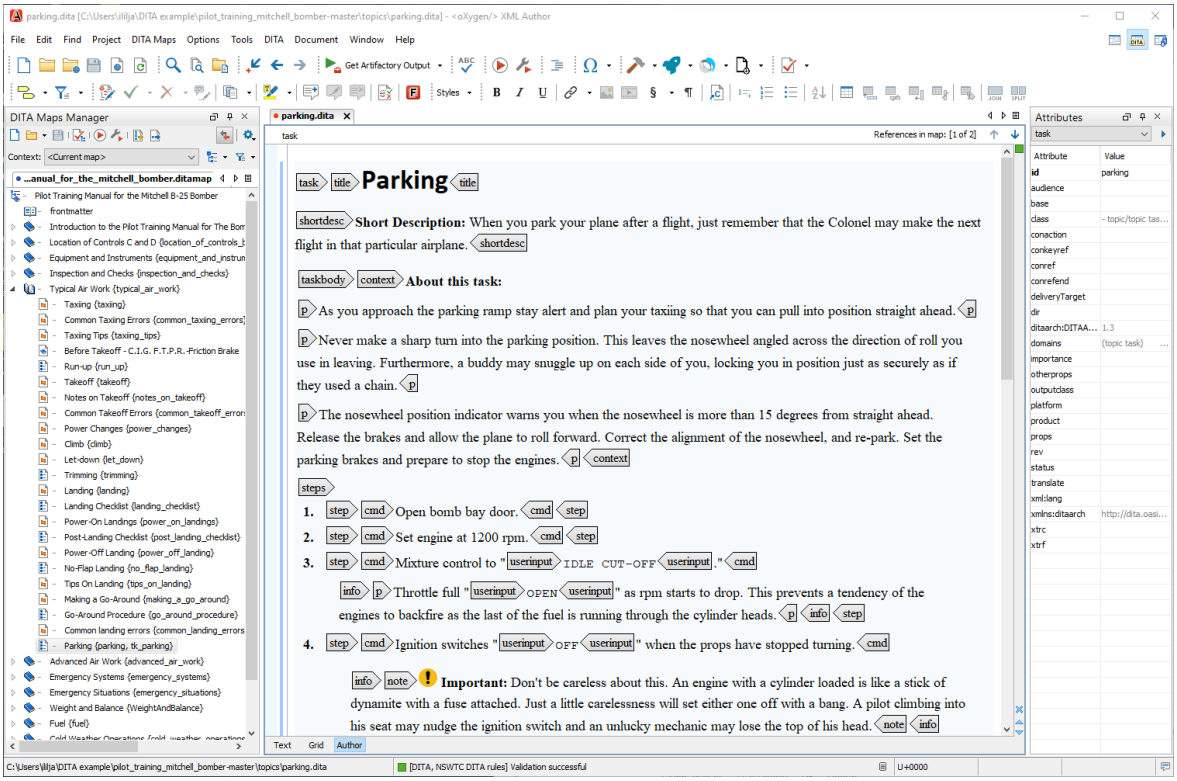


Figure 1. Author mode view of oXygen (Source: Schengili-Roberts 2020).

In addition, the writer can choose whether they want to view or hide the XML tags – in Figure 1, the tags of the XML-authored content are displayed fully but without attributes that are listed

on the right. Alternatively, the writer can choose the Text mode shown in Figure 2 where the same content as in Figure 1 is displayed as XML source code, and the syntax is highlighted with different colors and indentation to distinguish the XML structure.

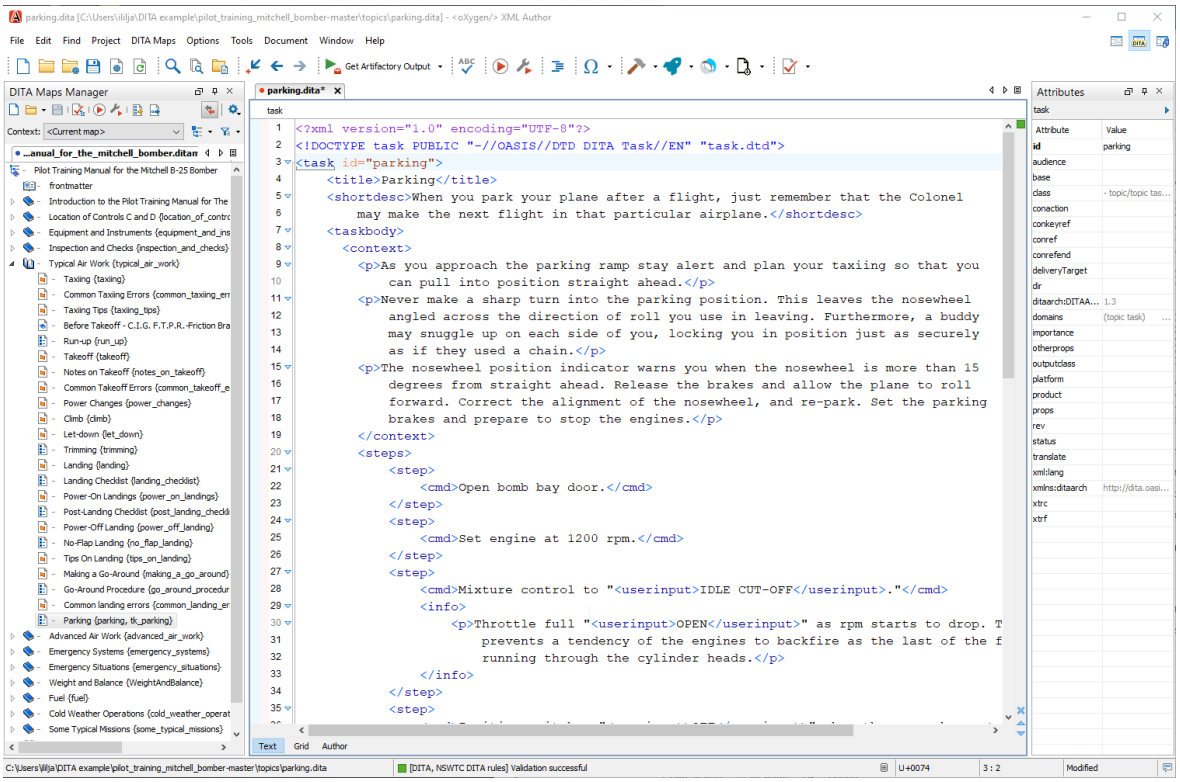


Figure 2. Text mode view of oXygen (Source: Schengili-Roberts 2020).

When the XML content is finished and validated against its DTD, the document can be published. The output format and appearance are determined in a separate process. Without that process, the XML content is only information that is tagged by using semantic markup but cannot be used for any other purpose than describing the information it includes.

Tools that are specified for producing XML-based content also enforce writing valid XML. For instance, if a file does not follow the syntax rules of XML or the DTD, oXygen will not process the file; instead, it will prompt the user to fix the errors. In general, this feature ensures that XML documents are valid and can be processed reliably by computer software. If a writer tries to insert an element in a position that is against the XML specification or the DTD rules, oXygen first alerts of the invalid position, and if the writer persists on using that position, oXygen rejects the insertion position and moves the element to a valid position in another location within the file that is allowed by the DTD. In Figure 3, oXygen warns that inserting an <option> element immediately after the <title> element is not possible. Figure 3 also shows how oXygen's drop-down menu of semantic elements instructs the user on how to use each element by including

an explanation of their intended use. How the DITA elements should be used has been defined in the DITA Language Reference chapter of the DITA standard (OASIS 2015), and oXygen displays the proper definition in the tooltip when the drop-down menu is opened.

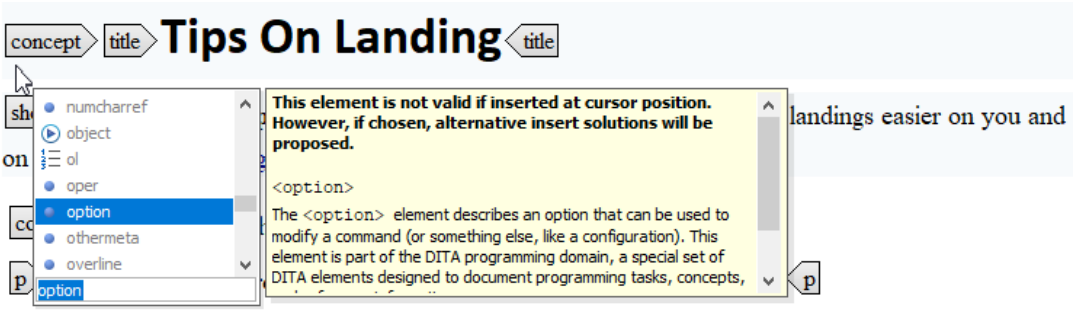


Figure 3. Example of an invalid insertion position. (Source: Schengili-Roberts 2020).

According to Self (2011, 225), XML transformed the way of storing knowledge and data when various industries, governments and other interest groups could create their own standards, such as the *Chemical Mark-up Language* and *Scalable Vector Graphics* to suit their needs. This progression extended to the technical documentation field where new forms of structured writing approaches emerged thanks to XML and the open-source movement (ibid.). XML has gained popularity among practitioners because of its many capabilities regarding structured writing, including the separation of content and form; structure and rules for describing content via a specific, customized semantic markup; and the standardization of documentation solutions (Priestley et al. 2001, 352–353). For organizations, XML can mean cost-savings in terms of localization and content creation, as well as improved quality and consistency thanks to the rigid structure it imposes (Katajisto 2020a). One of the popular XML languages in the technical documentation field is DITA XML.

2.2 DITA XML

As was stated earlier, DITA is one example of an XML language. But what kind of an XML is DITA? As Self (2014) summarizes, in addition to being XML, DITA is

- a structured authoring platform
- a topic-based information architecture and
- a writing methodology.

The following subchapters address these three perspectives of DITA after a brief introduction to its basics and origin.

In early 2000s, DITA XML was created within the technical publications department at IBM, where pioneer knowledge of multiple document formats was applied to the development of an XML-based document approach (Self 2011, 221). According to Schengili-Roberts (2017, 205), much of the design work originated from IBM's internal guide for developing technical content that emphasized minimalism (see subchapter 2.4), separating conceptual information from procedural instructions, and *chunking*, which is the practice of grouping information into units by content and labeling those units by using the @chunk attribute.

From the very beginning of DITA, it was intended to be published as an open standard due to the possible benefits that collaboration could offer (ibid.). In 2004, the standard was donated to OASIS[1], who in 2005 approved Version 1.0 of the DITA specification as an open standard to be adopted by the technical writing community. Since then, OASIS has maintained the standard, and at the moment of this research, the analyzed four components use Version 1.3 of DITA (released in 2015), which is the latest approved version. Along with DITA XML, IBM also developed the DITA Open Toolkit (DITA-OT), a publishing engine for DITA-authored content whose output formats include XHTML and HTML, PDF, and Markdown among others. Similarly to DITA XML, DITA-OT was released to open source in 2005. After being published, the DITA standard spread to numerous early adopters within various business sectors, especially to high-technology organizations such as Adobe and Siemens Medical who, via their access to more advanced tools, could fully exploit DITA (Schengili-Roberts 2017, 205). Consequently, members of their writer teams carried the accumulated best practices in technical documentation to other organizations over time (Schengili-Roberts 2014; 2017, 205). Schengili-Roberts (2017, 201) estimates that DITA is the fastest-growing standard for structured content. According to his informal research of DITA-practicing organizations, the computer software sector forms a clear majority of all DITA-using organizations, which is explained by the fact that this sector comprises many of the early adopters of DITA (ibid., 205–209; 2021).

The acronym *DITA* stands for the main principles of the standard. *D* pays homage to Charles Darwin's evolutionary adaptation theory whose principles of specialization, adaption and inheritance are incorporated in DITA (Self 2011, 219). *IT* is for Information Typing, which refers to the semantic structure and categorization of information according to its purpose

[1] OASIS, the Organization for the Advancement of Structured Information Standards, is a standard body that works to advance the development of open-source software and standards globally (OASIS 2022).

(Self 2011, 220; Schengili-Roberts 2017, 202). *A* indicates that DITA is more than an XML standard: it is an Architecture, a workflow, a methodology and a philosophy (Self 2011, 220), and that the standard is extensible (Schengili-Roberts 2017, 202).

As was described above, DITA is an end-to-end, open standard for creating and managing written documents. DITA is also a semantic, XML-based markup language that separates content from form by using document semantics as the basis for markup. Semantic markup is the practice of writing and structuring content by using XML elements based on what the meaning of the content is rather than how that content should appear when it is published. How semantically marked-up content is eventually displayed is not determined in the source file, but instead at a later stage outside the authoring process (Self 2011, 225).

2.2.1 A structured authoring platform

One of the most prominent principles of DITA is that it is a platform for authoring *structured content*. In XML, structure cannot be measured merely by whether a document is methodologically divided into sections or chapters – structured authoring covers a myriad of other aspects, too. According to Self (2011, 225), the following is the "modern definition" of structured authoring:

> A standardized methodological approach to the creation of content incorporating information types, systematic use of metadata, XML-based semantic mark-up, modular, topic-based information architecture, a constrained writing environment with software-enforced rules, content reuse, and the separation of content and form.

The key information in this definition is that structured content is modular and content is separated from form. Self (ibid., 223–224) defines *modularity* as the technique of compiling large, complex entities in smaller, self-contained pieces – Self uses an everyday example of a modular couch that, instead of being one large piece, is built of four pieces that can be arranged in various ways to form different couch configurations. Similarly, documents can be written modularly, and as stated by Self (ibid.), modular documentation can be produced and localized more efficiently and is easier to maintain. However, modularity equals a larger number of files to manage, which in turn requires effort for content management purposes.

As Self (ibid., 225; 2014) remarks, structured authoring includes discarding form. This is an inseparable characteristic of DITA as it does not have a native presentation format – DITA tags simply label what the information is, not how it is displayed. Whereas style-based markup uses

different heading styles, fonts and formatting elements to describe a document, the DITA approach separates content from its form by using semantics instead of presentation as the basis for markup. Self (2011, 229) defines form as the format, style and presentation of writing. He states that while the writer is not responsible for form, their job is to separate content by using semantics to identify each element of the document. Similarly, Priestley et al. (2001, 353) advise against mixing presentation structures with content-specific markup. They also argue that refraining from using formatting mechanisms such as fonts and page breaks is not enough, because if the content imitates a book-like structure, the published online version will still look like a book. This, in turn, could confuse users who are expecting a different type of structure.

Perhaps the most important trait of separating content from form through semantic markup is the fact that it enables *single-sourcing*, which is a methodology for creating, using and reusing documents. As Ament (2002, 1–6) explains, single-sourcing means developing modular content in a single database, assembling the content into different formats, such as printed manuals and online help systems, and reusing the content. Reusing is cost- and time-friendly because it eliminates duplicate work. In addition, updating content takes less effort because when content is reused via single-sourcing, any edits to the source text need to occur only once as the changes are extended to all publications (Self 2009, 254). Ament (2002, 1–6) continues to argue that single-sourcing also improves the quality of the documentation. According to him, it makes document usability "an all-or-nothing" issue because usability guidelines must be incorporated and implemented in the documentation standards for the methodology to succeed (ibid.). After assembling the document, it is converted into the desired output format depending on the needs of the organization. Ament (ibid., 188) lists three types of single-sourcing tools that are needed in the documentation process: authoring tools for producing the content (e.g. oXygen), conversion tools for automatically rendering the content from one format to another (e.g. DITA-OT), and content management systems (e.g. Git) for managing large amounts of content.

Schengili-Roberts (2010) argues that since DITA is an XML-based markup language, writing in it necessitates both writing and thinking in a structured manner. In DITA, different pieces of information have their own categories according to which they are arranged in the document. Similarly, Schengili-Roberts (ibid.) describes structured authoring as an approach for presenting the information that the end user needs to know when they need to know it. For example, certain information in a procedure, such as mandatory settings that must be configured before executing the following commands, must be presented before instructing the user on the

first step – this type of information that the user must know or must act upon can be categorized as prerequisite information, and therefore it should be included in the corresponding <prereq> ("prerequisite") element. On the other hand, information about what the user should do after they have completed the procedure should logically be presented at the end of the procedure – for this purpose, the <postreq> ("post-requisite") element should be used. By categorizing information in this manner, both computers and humans can identify the purpose of the information. However, it is humans who must recognize where each piece of information in a document belongs or whether a piece should be moved to another location – or perhaps be omitted from the whole documentation set. By encapsulating information in this manner, procedures and other types of content become self-contained in the sense that they convey all the information that a user needs to know about that particular subject, saving the user from the trouble of having to look for information elsewhere (Schengili-Roberts 2010).

2.2.2 A topic-based information architecture

A document authored in DITA consists of *topics* or *modules*, which are the basic units of DITA content and content reuse. A topic is a chunk of information that is stored in its own file. The DITA DTDs declare that at a minimum each topic must have a title, but often topics also have a body of content with one or more elements depending on how specific or detailed the topic should be. The DITA specification (OASIS 2015, 26) describes DITA topics as follows:

> DITA topics consist of content units that can be as generic as sets of paragraphs and unordered lists or as specific as sets of instructional steps in a procedure or cautions to be considered before a procedure is performed. … Classically, a DITA topic is a titled unit of information that can be understood in isolation and used in multiple contexts. It should be short enough to address a single subject or answer a single question but long enough to make sense on its own and be authored as a self-contained unit.

According to Hackos (2006), topic-based authoring began to spread within the technical communication field when online help systems appeared in the mid-1990s and writers acknowledged that they could not split existing user documentation – typically books – into help topics that would render each heading level a new help page. The modularity approach originates from *minimalism* (see 2.4), which is a framework for designing instructions that was popularized by John Carroll in the early 1990s. Today, the popularity of topic-based authoring can be explained by the flexibility, effectiveness and user-friendliness of succinct topics as opposed to lengthy, linear manuals. Hackos (ibid.) also notes that topic-based authoring continues to be popular because it makes content more flexible as the information is readable with or without context. Additionally, the approach can also decrease time to market and

increase efficiency because content can be produced faster and for less cost, and topics can be reviewed and translated before the entire document is complete (Hackos 2006).

The objective of writing documentation in DITA is to create a web of information by authoring content in discrete topics, organizing them into logical collections and establishing links between related topics (Bellamy et al. 2011, ch. 1). Even though DITA does not enforce any particular writing practice, it is designed to support authoring, managing and processing topic-based content that is designed to be reused (OASIS 2015, 27). To do so, topics should be *self-contained*, which means that the amount of information they carry suffices to cover one idea while also being understandable without requiring the user to read additional context from other topics. All topics do not need to be self-contained units of information; some topics may contain only titles and a short description and serve primarily to organize subtopics or links to topics for various purposes, such as information management or authoring convenience (OASIS 2015, 26). The self-contained topics should not live alone either: topics need to have a "home" of their own in a larger entity, an organized collection of topics (Bellamy et al. 2011, ch. 1). In DITA, this home is the *DITA map*, which is a structure for defining which topics constitute the collection of topics that will be processed into the output publication as well as the topics' sequence, hierarchy and linking relationships (Self 2011, 29). In a DITA map, topics are <topicref> ("topic reference") elements, which are essentially links to the individual topics.

Similarly to many other elements, topic references can be *nested*, which means enclosing elements by other elements. Nesting also creates a parent–child hierarchy structure: the *parent element* is an element that contains one or more elements that are called *child elements*. The DITA map essentially defines the table of contents for the topic collection by nesting topics. A DITA map can also include other maps that can comprise user documentation topics or resource-only assets, such as glossaries. If this thesis were authored in DITA XML, chapter 2 "DITA in technical communication" would be the parent element of subchapter 2.1 "Markup languages" in the DITA map. The first lines of a simplified DITA map would look like this:

(5)
```
<map>
    <title>Using and advancing DITA XML – A case study of evaluating DITA-authored customer documentation</title>
    <topicref href="introduction.dita"/>
    <topicref href="DITA_in_technical_communication.dita">
      <topicref href="markup_languages.dita"/>
      <topicref href="DITA_XML.dita"/>
    </topicref>
```

DITA topics can be used in any number of maps, and any number of topics can constitute a DITA document – for example, a parent topic with nested child topics can represent an entire deliverable (OASIS 2015, 26). However, the potential of DITA is most effectively realized when each topic is stored in a separate XML document and by organizing them into maps based on how topics are combined to form deliveries (ibid.).

Furthermore, the topic-based, modular information architecture of DITA is closely linked to information typing, which is the practice of identifying different types of topics to distinguish between different types of information that answer distinct questions that the users have, such as "What is...?" or "How can I…?" (ibid., 24–28). Typically, information types have a set of elements that can only be used in that specific information type. The default DITA content model defines three base information types: *concept*, *task* and *reference*, each of which inherit characteristics of the most general information type, *topic* (Self 2011, 11). The list was amended by the *troubleshooting* information type in DITA 1.3 (Thomas 2014).

Table 1. The four DITA information types.

Information type	Use case	Note
concept	For documenting conceptual, descriptive or overview information, such as background information that the user must know before they can successfully work with a product or within a process. Answers the question "What is?"	Concept topics can be broken into sections, but they are primarily comprised of simple paragraphs and unordered lists.
task	For documenting the steps of a particular task or procedure, or to document the stages within a high-level process. Answers the question "How to?"	Task topics are more strictly structured than concept and reference topics with corresponding elements that are specific to this topic type, such as <prereq> and <taskresult>.
reference	For documenting fact-based, supporting information for performing a task, such as parts lists, product specifications.	Reference topics often present information in lists or tables. Information is not explained on a deeper level.
troubleshooting	For documenting corrective action information, including the description of the condition that the user might want to correct and instructions for remedying it.	Troubleshooting topics follow a rigid schema with corresponding elements that are specific to this topic type, such as <cause> and <remedy>.

The original three base DITA information types are also reflected in the information type model that is presented in the standard for the preparation of information for use of products, SFS-EN IEC/IEEE 82079-1 2020 (71). Each DITA information type is complemented by a corresponding DTD. The DTD schema of each information type varies – for example, the task DTD declares that tasks can contain procedural steps (<step>) that must be within a <steps> element. The concept DTD instead does not allow using the <steps> or <step> elements.

As Katajisto (2020c) states, information types do not require DITA, but using DITA makes writing more effective as the writer does not need to reinvent how information is organized. For writers, information typing serves as a guideline that provides a structure for conveying content to the user: concepts are for providing information of product features, while tasks are for step-by-step instructions on how to do something (Schengili-Roberts 2017, 203). This way, the writer can follow the set rules and focus on the content itself instead of how the content should be structured. If the information is conceptual, they should create a concept topic; if the information concerns a procedure, they should create a task topic. The authoring tools cannot decide which topic type should be used – they simply provide the appropriate structure and define the elements for the chosen topic type.

Choosing the topic type according to the content is important because information types should not be mixed in a topic-based writing environment. Bellamy et al. (2011, ch. 1) explain that "when it is time to adjust the valves on the motorcycle, users do not want to read a novel. They want to open the motorcycle manual, find that one specific task, and move on." In that example scenario, the appropriate DITA topic for this problem would be a task topic with step-by-step instructions for adjusting the valves, possibly accompanied by any necessary notes or warnings, or perhaps related pre- or post-task information. What the user would not want to read is conceptual information about the valves because when reading the task, they have a real-life task (adjusting the valves) that they want to complete, presumably as quickly as possible, so that they can move on to something else. Including any information that is not strictly related to the task or subject in question could distract the user who wants to find the necessary information and move onto the next task. In fact, confusing product information is a major source of frustration for consumers and skilled workers (SFS-EN IEC/IEEE 82079-1 2020, 71). In addition to aiding the users navigate and consume documented information, separating information into discrete topics facilitates creating new information consistently, eliminating unimportant or redundant information, and identifying common or reusable topics or pieces of content in the documentation (Bellamy et al. 2011, ch. 1).

2.2.3 A writing methodology

Even though DITA can be described as a tool for technical documentation, it is more of a method that covers various aspects of writing. The required changes in the ways of writing are closely connected to Schengili-Roberts' (2010) argument that authoring in DITA is very

different than traditional writing, because the structured nature of DITA necessitates the writer to think differently than when working with regular, linearly written documents. This statement is supported by Ament (2002, 5) who states that linear documentation is almost impossible to reuse, which is one of the main incentives of DITA. Linear writing refers to the idea of writing information by arranging it so that is flows logically from one subject to another, while also forming connections and dependencies between the subjects. Once the text has been written, the order of the subject remains stationary, and the reader is expected to follow it closely. Books are a typical example of linear texts. However, one of the motivations of DITA is to move away from the old-fashioned narrative, book-like structure that has been used in technical documentation. As Bellamy et al. (2011, ch. 1) explain, books are a suitable option for some occasions, but they are not the best medium for delivering targeted technical content to users with real-world tasks to complete. This is because books, like other linear texts, assume that the reader starts reading from the start page by page, proceeding towards the final page – but today, this is not how a typical reader of customer documentation operates. On the contrary, Priestley et al. (2001, 354–355) point out that since different users have different needs, they likely retrieve information by entering the document at different places. In other words, a technical document does not have a clear-cut starting page, which is why each page should be treated as if it were the first page that the user reads.

Priestley et al. (ibid.) also argue that writing in a linear structure forces writers to arrange subjects into order even though there is no single correct order because the subjects could fit in different places and different collections. Writers often pick the order based on how to make the text flow better and avoid repetition and cross-referencing, but as a consequence, the subjects are dependent on each other (ibid.). This, however, may not be the most effective choice from the end users' viewpoint, which is illustrated by Hackos (2006), who states that "[r]eaders have goals, and they want to achieve those goals as quickly as possible." The user is highly likely to be in a hurry, and for some users, documentation is their last resort to which they turn only when they are stuck (van der Meij and Carroll 1998, 42). In a typical scenario, the user accesses the user help (whether in electronical or printed form), searches for the instructions for a particular problem, scans only that specific content and then quickly moves on to do something else. The user will not continue to read the following subject of the document because it happens to be the next chapter – they will return to the instructions only if they need them. The structured, topic-based writing methodology of DITA was designed to

comply with this type of use, but to implement this, writers may need to rethink their way of writing to reflect the DITA principles.

The differences between writing linear documentation and writing structured text in a DITA environment also apply to seemingly minor details. Self (2011, 128) reminds that when context is removed from the content, subjects in the source file may not appear in the same sequence in the output because some items might be deleted in the publishing process (see 2.3). To illustrate, Self (ibid.) argues that if the order and number of items in a list cannot be predetermined, it would be counter-productive to insert the word "and" to the penultimate item or to add a full stop to the last one, even if the stylistic or grammatical convention would be to do so.

The end users of documentation are not the only ones who benefit from structured authoring in DITA – based on my experience as a new team member, authoring properly structured documents that use the correct semantic elements is likely positively reflected in the day-to-day work of writers, too, because when a new writer joins the team, it might help them to understand the content better when they can view its purpose and structure instantly by looking at how it has been marked up and organized. For example, if a task topic instructs how the user should operate a user interface (UI), the procedure can be difficult to perceive for someone with limited experience of that specific task or that specific UI. Instead, if the topic has been written in a logical manner and if it uses semantic elements to describe what the things in the content are (such as <uicontrol> for UI controls or <userinput> for input required from the user), it can help the new writer gain a better understanding of the content of that task topic. The four components whose documentation is analyzed in this study represent complex software, and therefore the documentation contains numerous references to how the software should be operated. Using the corresponding DITA software elements would also aid the writer to understand the product when program code, for example, is distinguished from the main text flow as it is tagged with code-representing elements. Thus, following the key principles of DITA benefits both the target audience and the creators of technical documentation.

2.3 Main features of DITA – inheritance, specialization and reuse

In addition to the abovementioned viewpoints, DITA includes many other features. Among others, the principles of inheritance and specialization (the *D* in DITA) are an essential part of the architecture's design. This is exemplified in how the four base information types (concept,

task, reference and troubleshooting) are called *descendants* of the proto information type, topic, from which they have evolved. The base information types have *inherited* the common base structure and characteristics of their ancestor (Self 2011, 226). The DITA architecture is based on this process of evolving that is called *specialization*, which means forming new entities from pre-defined ones. The ideas of specialization and inheritance borrow from the Darwinian theory of biological evolution, according to which species change over time, thus forming new species that share a common ancestor. New species evolve to better adapt to their environment – similarly, new DITA specializations are created so that they would better serve the nature of the content. In DITA, the base DITA components (such as the <topic>, <body> and <example> elements) are the species ("ancestors") which are specialized into new ones. Self (ibid., 227) argues that specializing topics or elements is easy because you only need to define how the topic or element differs from its immediate ancestor. All DITA information types and elements exist in a parent–child relationship, and specializations inherit the properties of their parents (Schengili-Roberts 2017, 202–203). The inheritance history of DITA elements is recorded in the DITA Language Reference. For example, the base topic <topic> is the parent of the specialized task <task> topic type, and <taskbody> is the specialized task topic equivalent for the <body> element. The generic topic type is used for untyped topics (OASIS 2015, 182), that is content that does not fall into any specific information type category. However, the downside of writing untyped content is that it leads to unstructured writing. Additionally, only typed topics contain built-in features that enable collecting topics by category (e.g. by information type) in the navigation systems accessed by users (ibid., 183).

When should an organization specialize DITA? If the default structure of a topic cannot fulfill the needs of the organization, it can specialize topics, elements and attributes by defining unique, organization-specific entities according to DITA's specialization rules. Specialization is done by using existing definitions (elements, topic types and so on) as a basis for new ones. By default, the new definitions are processed similarly to their ancestors, but they can be specialized to be processed differently as well. Although specialization is often viewed as an arduous task, it might benefit organizations who want their documents to follow an organization-specific structure or to contain specific metadata (Self 2011, 232). At the same time, organizations should be wary of specializing without an actual need (ibid.). The business unit whose documentation is analyzed in this thesis has decided not to specialize because of changing documenting tool environments, which are beyond the documentation teams' control,

and migrating into or from a specialized DITA environment is arguably more demanding when compared to a default, non-specialized DITA scenario.

Another key characteristic of DITA is reusing content. Essentially, reusing content is the practice of utilizing existing pieces of content in the development of new documents. It means writing the text once and then reusing it whenever needed. Effective reuse can result in improved efficiency and consistency as well as reduced documentation costs, especially those related to updating, reviewing and localizing content (Bellamy et al. 2011, ch. 10). The copy-and-paste function is probably the most well-known reuse mechanism, but it does not suffice for writers as it requires a great amount of both time and manual work and is prone to human-error. Reducing the need for manually copying and pasting content from one place to another is one of DITA's core principles (ibid.), and to accomplish this goal, DITA offers various ways of reusing content.

The reuse technique used in DITA is called *reuse by reference*, which coupled with single-sourcing allows updating the single source and having the updates picked up automatically wherever the source is used (Priestley et al. 2001, 357). Reused content can either be embedded as text in the target topic, or it can be inserted as hypertext that can be clicked to access the link target. Any element can be referenced; the only prerequisite is that the referenced element must have an @id attribute, which is an identifier for DITA elements. Reuse by referencing also reduces some of the typical risks that documentation teams face; Bellamy et al. (2011, ch. 10) describe the "never-ending" cycle of changing product names or user interface labels, for example, in product releases where time is typically of the essence. They note that product development teams cannot be demanded to stop making last-minute changes, but if writers are reusing frequently changing content by reference, they are able to rapidly update content from the single source without having to edit each occurrence manually.

In DITA, content can be reused on two levels, the first of which is topic-level reuse. Reusing topics simply means using individual topics in more than one context either within the same or another DITA map. Topic-level reuse is described as the primary means for reusing content at the first level of DITA adoption (Priestley and Swope 2008, 5). Topic-based writing is a prerequisite for topic reuse because only independent, context-free topics can be reused. Even if the documentation consisted of stand-alone topics, reusing them as they are is not always possible due to differences between products or documents; this problem can be mended with the *conditional processing* mechanism. Conditional processing means applying metadata

attributes in DITA content for filtering fragments of content during the transformation process (when source files are rendered into the output format) so that the content can be used in various contexts (Self 2011, 189–190). The filtering rules are defined in a separate DITA values (ditaval) file. In practice, conditional processing allows reusing and single-sourcing content with varying values for certain attributes, which Self (ibid.) refers to as *condition attributes*[2]. Any element regardless of its size can be assigned various attributes that can be processed to either exclude, include or highlight that piece of information when the content is published. In example 6, a clause within a <p> ("paragraph") element contains two phrase <ph> ("phrase") elements with differing @platform values (either "windows" for Windows users or "mac" for Macintosh users) depending on the operating system of the user.

(6)
```
<reference>
  <title>Supported software</title>
  <refbody>
   <p>The application supports <ph platform="windows">Windows 11</ph> <ph
   platform="mac">macOS Monterey</ph>.</p>
  </refbody>
</reference>
```

If the reference topic in example 6 were published for Windows users, the content would be filtered so that any element whose @platform value is something else than "windows" is excluded from the output. The end-result of the paragraph <p> would be "The application supports Windows 11." The same logic could then be applied to Macintosh users or any other users – if the application were to support Linux, the writer would only need to add another <ph> with the @platform attribute set as "linux". This way, the same topic titled as "Supported software" can be reused for different target audiences instead of writing and maintaining separate topics for all possible user scenarios. Thanks to specialization, organizations can create their own attributes as well to make their content more reusable. However, Self (2011, 193) warns that without caution, filtering when single-sourcing may result in invalid DITA. Writers need to ensure that XML remains valid even if all conditioned content is filtered out during the publishing process.

The second level of reuse occurs on the element-level. This refers to reusing block or inline elements from other topics without having to create a new topic with slightly different content. Taggable block and inline elements are reused within topics via the *conref* (content reference),

[2] The DITA condition attributes include @audience, @platform, @product and @otherprops.

keyref (key reference) and *conkeyref* (content key reference) mechanisms. As Self (2011, 172–173) explains, content referencing is realized via *transclusion*, which is the ability to identify a piece of information by using a simple naming structure, or a reference. A mundane example of transclusion is how the content of Wikipedia pages can be included within other pages by reference, and the same content is included in multiple documents without having to edit the documents separately (Wikipedia, 2022a). Self (2011, 172–173) describes how the transclusion mechanism works in DITA:

> A content reference is specified by entering the address of the content to be re-used in the conref attribute of the element into which it will be included or *transcluded*. The conref attribute value has the following syntax: `topicfilename#topicid/elementid` where *topicfilename* is the file name of the topic containing the content to be transcluded, *topicid* is the id attribute of the topic, and *elementid* is the id attribute of the element to be re-used.

The referenced source content is added to the target topic where it cannot be edited. The referenced content is maintained in the source topic, and whenever the writer updates the source, all the target topics containing that specific piece of content are updated automatically as well. This is very efficient in situations such as when a spelling error is fixed, a code string is updated or when a product name changes. In essence, content references are practical whenever information that is repeated in multiple locations needs to be altered: writers do not have to manually search and rewrite every instance that has changed – of which there could be hundreds in a large document – as updating the single source updates all content. The downside to the conref mechanism is that it functions by referring to a physical location in the file system, and if that target location is missing (if the file was moved, deleted or filtered out), the document cannot be transformed or published. In addition, Bellamy et al. (2011, ch. 10) note that conrefs create dependencies between files, which can limit the reusability of content. They also point out that writers depend on the work of other writers because referenced files must be complete and available when the document needs to be transformed and delivered. As a solution, they propose using designated reuse files (ibid.). Furthermore, the business unit recommends referencing content by using *keys* instead as they are simply ignored in the transformation process if the key's target is unavailable. In addition, they enable more efficient content reuse.

A key is a name for a resource that is referenced with either the @keyref or the @conkeyref attributes (OASIS 2015, 66). The referenced entity is the key name and its given value. The values of keys are defined in a <keydef> ("key definition") element in a separate topic or DITA map which is included in the document DITA map. This means that keys are scoped to the respective DITA map in which they exist. Keys increase the reusability of topics because a

topic with reuse potential could include information that varies depending on the product in question. For instance, if a referenced key in a topic were named "product_name" and that topic would appear in two different DITA maps, it would be separately configured in map 1 for product A and map 2 for product B. As a result, the value of the key would be "product A" in map 1, but in map 2, it would be "product B". If the products include similar content, whether it be topics or elements, that content can be written once and then reused by using the product_name key whenever the product name must be written. These types of referenced entities that are reused by replacing their value when the information is published are called *variables* (Self 2011, 178–179).

For writers, reusing content means that they do not need to rewrite information that has been documented before, but it also means that they must write content consistently so that it can be reused. Another important aspect of writing for reuse is avoiding transitional information, which can be phrases such as "In the previous section, …" or "At the beginning of this document, …". Referring the user outside the topic with these types of linguistic devices is not practical in topic-based writing firstly because writers cannot know the order in which topics are assembled in a document, nor should they refer to topics without providing a link[3] to them. As a rule, well-designed DITA topics are context-free and can be reused in other documents without revision of their content as long as writers are careful to avoid unnecessary transitional text (OASIS 2015, 27). Moreover, Self (2011, 126) notes that information typing and using specialized topics is the method in DITA for separating content that may otherwise have been grouped together, and this separation reduces the need for transitional text within topics.

To summarize, DITA supports "a maximal implementation" of content reuse (ibid., 171). Schengili-Roberts (2017, 215) states that when compared to other available documentation standards, DITA differs from them with its ability to reuse content at not only the topic-level but also granularly, meaning that even the smallest fragment can be reused. He also argues that these two levels of reuse have a positive impact on DITA's popularity within the field because they bring the benefits of improved consistency, lowered localization costs and the increased efficiency of writers. To achieve these benefits, writers must be aware of which type of content is reused within their team and how reuse should be implemented. In addition to having a reuse strategy, writers also need to write for reuse, which is related to the minimalist approach.

[3] In DITA, the <xref> element is used for creating links to DITA topics or a specific element within a topic.

2.4 DITA and minimalism

This chapter discusses the minimalism principles from the perspective of DITA. The word *minimalism* is often associated with maximal scarcity and simplicity. Despite the semantics of the word, minimalism in technical communication does not equal simplifying, shortening or reducing. Instead, minimalism is a user-centered theory that was promoted by John Carroll and others in the 1990s. Minimalism is often described as an approach that presents the reader with the smallest amount of information that is necessary to achieve the reader's goals, thus shifting the focus from the features of the system to the needs of the reader (Self 2011, 224). The starting point for Carroll was to rethink the way documentation is presented to users – in his first book on minimalism, *The Nurnberg Funnell* (1990), Carroll discards the so-called systems approach to user guides, which emphasizes the features and workings of the system while paying little attention to the users of said system or the environments and contexts in which the users operate. According to Carroll (1990, 73–74), the disadvantage of the systems approach is that it treats user tasks as systematic constructions that can be laid out as perfected step-by-step instructions that include even the most basic tasks, such as moving the cursor across the screen to click something. While breaking tasks into steps is considered to be good practice in technical documentation, the same cannot be said about disregarding the users' skills, needs and motivation. Carroll (ibid. 74–75) argues that users should be presented the opportunity to do something meaningful by applying their pre-existing knowledge in the pursuit of their goals. In other words, users should be instructed when they need instructions or guidance, and they should also be encouraged to be independent and use their own intellect. The instructions are written for situations where the users have doubts or seek reassurance.

To illustrate how widespread minimalism is in the technical communications' field, the IEC/IEEE standard for preparing instructions for use for products concisely declares that "minimalism shall be applied" (2020, 86). In addition to being a well-known guideline in technical communication, minimalism is also a central design feature of DITA. Stevens (2018) argues that when used together, DITA and minimalism generate documentation that is concise, consistent, accessible and relevant. Nevertheless, Stevens (ibid.) clarifies that although the two practices complement each other, they are separate – DITA supports but does not guarantee minimalism, which is also why content that is converted into DITA does not automatically transform into minimal content. Minimalism in DITA must be enforced by writers, which requires understanding the minimalism theory that comprises four principles and a list of

associated guidelines that were introduced by van der Meij and Carroll in 1995. Stevens (ibid.) argues that these four principles are clearly represented in the DITA standard.

The first principle is to choose an action-oriented approach. It calls for offering meaningful real-life tasks to the user immediately so that they can learn by doing and thus maintain their motivation for learning, as opposed to first reading pages of instructions or background information. Van der Meij and Carroll (1995, 244) state that minimalist instruction is always action-oriented, and it should encourage and support user exploration and innovation. This is in line with Stevens' (2018) thinking who remarks that users do not seek documentation in order to find things they already know or can quickly figure out. Instead, they access the documentation to find more advanced things that they cannot immediately understand without the guidance of the documentation, which according to Stevens (ibid.) is what the documentation should focus on. Stevens (ibid.) describes minimalism as a subtraction whose aim is to focus instead of subtracting for the sake of subtraction – the documentation should consider the equation of what the users need to know and what they already know. Therefore, the end-goal is not to reduce the number of words or the length of the text but to give the users the best value by emphasizing their goals. Similarly, information that is unnecessary for the user should be omitted. Stevens' (ibid.) advice is to omit such information that the user already has or can figure it out on their own, especially if they do not need it in the first place and omitting it does not pose any major, long-lasting risks to either the user, their data, or the company itself.

Because the user wants to get straight to the point and start the task, they do not want to have to comb through excessive background information that they do not need to read. In DITA, the topic title should tell the user what the topic is about, and Stevens (2018) warns against the tendency of including information such as "To accomplish this task, follow these steps:" in tasks because DITA does not have an element for informing the user to follow the following steps. This type of an element does not exist, because DITA uses other devices to convey this information: the user already anticipates that if a topic title begins with a verb, it is likely a task, and when they see the numbered steps, they do not need to be told that the topic is in fact a task.

The second principle instructs anchoring the tool in the task domain. The word *domain* refers to the real world of the users. The guideline reminds that all tools and applications are mediums for addressing the users' end-goals, which is why the instructions should not reflect the tools, but instead the tasks the user wants to accomplish. Van der Meij and Carroll (1998, 28) remark

that this is obvious to the user but can "easily escape" the writer. Similarly, Bellamy et al. (2011, ch. 1) stress that the writers' goal is not to simply describe how the product works – they should help users accomplish their goals. But what constitutes a goal? Hackos (2012, 1) argues that the most common error in user documentation is focusing on using the user interface of a product instead of how the user can complete real work – although operating in the UI supports the user's work, clicking on an interface object is never the real goal. In both minimalism and DITA, recognizing this separation is essential, and the relevance of real-life tasks should be emphasized.

Thus, tasks are at the heart of customer documentation and DITA. This is also argued by Bellamy et al. (2011, Part 1) who add that "[o]ne of the revelations of minimalism is the task-oriented topic that focuses on the goals of the user." In DITA, the minimalism principles clearly prevail in the task information type, which may explain why the task information type is the most strongly typed one (Self 2011, 97). The strict DTD of tasks also discourages including information that is not directly related to the procedure (ibid., 224). According to Bellamy et al. (2011, ch. 2), task topics distinguish technical documentation for products, technologies and services from other types of texts. Therefore, it could be argued that DITA-authored documentation should be task-centered. The structure of the task topic is more specialized than that of the concept and reference topic types, and the unique task elements, such as <prereq> ("prerequisite"), <step> and <stepxmp> ("step example") provide an effective structure for procedures.

The third principle instructs to support error recognition and recovery. Van der Meij and Carroll (1998, 3) find that "software users make many mistakes" and correcting them may be time consuming. When humans and software collide, errors are bound to happen – but that does not mean that some of them cannot be prevented by means of documentation. Virtaluoto et al. (2021, 22) suggest preventing errors by using hints and by providing effective error prevention information near error-prone actions or when recovering from the error is difficult. Using concise, jargon-free language can also help mitigate mistakes (van der Meij and Carroll 1998, 35–36). Writers can embed hints in the task steps, or they can be included as separate notes or warnings depending on the severeness of the consequences of ignoring them. Naturally, usability testing and user interface design also play a crucial role in minimizing errors. Consequently, the application itself may be the best avenue for error prevention (ibid., 36), but the documentation must still cover troubleshooting. In DITA 1.3, the third principle is evident

in the task topic that allows nesting a <steptroubleshooting> element within a <step> or the equivalent <tasktroubleshooting> within the whole task in case the user can be expected to require remedial information. In addition to the troubleshooting elements, a new base topic type dedicated for this principle, the *troubleshooting* topic, was added to DITA 1.3 (Thomas 2014).

The fourth principle is to support reading to do, study and locate. This refers to the users' tendency to process user guides in an unsystematic manner instead of the page-by-page, linear way. Van der Meij and Carroll (1998, 42) argue that most users browse manuals and retrieve information based on their current needs, which is why the manual should be designed in a modular fashion so that sections are findable and as independent from the previous ones as possible. This principle also advises to write concise documentation as everything does not need to be explained (Virtaluoto et al. 2021, 22). As was discussed in subchapter 2.2.2, DITA's modular, topic-based approach was designed to support this type of usage.

Considering these four guidelines and the discussion in subchapters 2.2 and 2.3, it seems that minimalism and DITA share the interest in users, task-centeredness and effectivity in technical documentation. This sentiment is supported by Hackos who calls a certain type of audience "strong candidates for a minimalist approach to informational text" (1998, 151–152). This audience to which Hackos refers to as "double experts" consists of experts who are experienced in both the knowledge domain of their profession as well as in the use of standard software applications. This type of an audience needs just the amount of information that is necessary for them to start learning how to operate in a new software environment. Consequently, they do not need detailed procedural instructions for manipulating the user interface nor do they require descriptive information to be able to understand the primary purposes of the application. As Hackos observes (1998, 151–152), they already know the subject-matter and what they want to accomplish; what they do not know is how to do it with that specific software. The end users of the business unit's software, too, fall into the double expert category, which would indicate that the customer documentation should follow the minimalist principles.

Virtaluoto et al. (2021, 32) argue that minimalism requires the whole business organization to participate in implementing the approach which includes conducting usability testing, training writers and information designers, but also adopting a new mindset that integrates minimalism into the product creation process from planning to maintenance. They also stress that the entire research and development organization needs user information in order to produce usable products, and writers could be an invaluable asset in creating this information (ibid.).

Still, one minimalist writing aspect is in the writers' hands, namely that of choosing the simplest approach to semantic markup instead of complex solutions and workarounds. As Self (2011, 95) states, the simplest form of markup reflects the best practice. Similarly, Self (ibid.) notes that it is not useful to add semantic tagging to content which does not need to be tagged – sometimes the end-result could even change the semantic nature of the content. He uses the <note> element as an example: if a note contains one paragraph, tagging that paragraph as a as in example 7 below makes it semantically different than a string-only note. A simple note highlights complementary information whereas a paragraph is a self-contained unit that consists of a single idea. Thus, a single note should be plain text without containing a paragraph.

(7) <note>
This note has one paragraph only. Therefore, it should be plain text.
</note>

2.5 Benefits and challenges of DITA

This section briefly discusses how the benefits of DITA can also be challenging to realize in general. While the key DITA features are efficient and practical, enforcing and internalizing them requires having strategies and certain competencies as they deviate from some of the more traditional technical writing conventions and practices. DITA is not a product that an organization can purchase and put into use; it is a multifaceted XML implementation that needs to be *adopted* both in terms of the tools used and how technical documentation teams realize structured, modular writing. As a result, the DITA adoption process is neither fast nor straightforward.

When an organization makes the decision to migrate to DITA, they may need to convert all or some of their existing content to DITA or even transfer it into a new content management system. This depends on their current tool environment, the source format of their documentation and whether their content is marked up. Converting, which means processing the source content by applying DITA tags to it and validating the markup, may be an arduous task even though it can be automated to some degree. First, depending on the content, it requires a team of multiple people including writers, information architects, documentation tool developers and other experts who together evaluate how the content should be divided into topics, how the topics should be managed and how they should be arranged in DITA maps among other issues (Bellamy et al. 2011, ch. 11). Second, marking up the content might be

perplexing with the vast number of DITA elements if there are no clear-cut rules as to which elements should or should not be used in the documentation. Third, although multiple source formats can be automatically converted into DITA thanks to the shared XML properties instead of manual copying and pasting, some post-editing of the content is often required (Lyytikäinen 2020, 27). This is because DITA XML editors and programming scripts cannot immaculately arrange topics into a logical order, apply reuse practices or ensure that the appropriate semantic elements are used. Finally, Bellamy et al. (2011, ch. 11) warn that in addition to the source content, the output files are important, too: "Your perfect DITA topics won't be of much use if you can't deliver them to users." Of course, this applies to all source formats. DITA-OT includes a few default output options, but as their visual design is somewhat rudimentary, organizations who use DITA will need to invest resources into designing a more unique, visually appealing appearance that conforms to their brand.

Another noteworthy issue related to the initial steps of adopting DITA is raised by Lyytikäinen (2020, 25) who notes that implementing single-sourcing and DITA comes with costs that are not restricted to software or tools – adopting a new approach to writing also affects how people work, which in turn necessitates training. As was mentioned earlier, DITA is more of a methodology than a tool, which means that to be able to use it correctly one must understand why the methodology is what it is. This, however, can be challenging because shifting from a linear, book-oriented way of writing into a topic-based, modular approach requires changes in the writers' ways of working. The shift may require a substantial amount of training, which may be accompanied by a steep learning curve and resistance to change (Katajisto 2020a). One of the likely hindrances to arise is that what you write is not what you see because DITA separates content from form, and authoring is not done in a what-you-see-is-what-you-get (WYSIWYG) word processing tool, such as Microsoft Word. As Self (2011, 230) notes, new DITA authors can have difficulties with how semantic authoring differs from style-based authoring. Self (ibid., 225–226) coined the term *what-you-see-is-one-option* (WYSIOO) to illustrate that what the DITA author sees may be completely different from what the user sees when they read the document. In brief, DITA determines what the context is, but the appearance of the content is not a part of that process; DITA is not meant for defining the appearance of content output, nor does it have sophisticated mechanisms for specifying it. Therefore, writers do not produce the delivery format output and cannot control it via writing DITA. Instead, deliverable documents' appearances are defined in separate stylesheets which are not a part of the DITA document (see 2.1). The presentational form of a document can even be unknown to

the DITA author (Self 2011, 230). Saunders (2010) argues that "ever since WYSIWYG desktop publishing software and word processors became widely available, writers have been evaluated on how good their work looks" with a heavy emphasis on a beautiful, book-like appearance, which he estimates to also be a highly enjoyable part of some writers' profession. Still, authoring in DITA does not translate into dismissing output processing; it simply means that someone else than the content writer will be responsible for the design of the output format or formats, while the DITA source documents are the archive format in which the source information is stored for further processing.

According to Saunders (ibid.), in industries where one would use DITA for technical documentation, the business case is not for "pretty" but for fast, correct and complete information as well as consistent presentation. Saunders (ibid.) underlines that this disruption of emphasizing good semantic tagging, clarity of presentation and maximized reuse instead of appearances can cause some writers to feel at unease, which could result in writing teams being "determined to turn DITA into a very bad word processor". In practice, this would mean (over)using non-semantic DITA highlighting elements, such as <b> for bold, <i> for italics and <tt> for monospace, or inserting empty block tags, such as paragraphs <p>, to affect the appearance of the output. Highlighting elements should only be used when a more semantically appropriate element is not available (OASIS 2015, 293–297). Self (2011, 125) warns that "if you find that you are deliberately coding your DITA topics in such a way as to achieve a particular output effect, you are without doubt taking the wrong path". Self (ibid.) continues to argue that writing for output should be avoided because the topics must be "delivery-agnostic" and free of as much context as possible, including how the information is going to be used, presented or sequenced. As Stevens (2018) explains, tagging content with highlighting elements for emphasis should also be avoided to prevent overstimulating the user's attention. Also, if a topic is authored according to the principles topic-based writing and minimalism, it should include only the required amount of information. In that case, all content is important, and formatting some information differently to highlight its importance should be pointless.

The possibility of reusing each piece of content is one of DITA's most distinctive characteristics and benefits. Reusing content saves resources, but in order to reuse content efficiently and consistently, a reuse strategy must be established. Priestley and Swope (2008, 8) point out that although there are many ways of reusing content with DITA, only such content that can be found can be reused – in other words, writers must have an established way of finding and

reusing information. Similarly, reuse works only if it is used appropriately and consistently (Bellamy et al. 2011, ch. 10). Naturally, all documentation teams must have a reuse strategy in place in order to maintain consistency in terms of reusing content; without an agreed strategy, the team risks endangering the quality and accuracy of content. Similarly, a flawed reuse strategy can confuse new writers, complicate the localization of the content and create disruptive dependencies (ibid.). Another challenge associated with managing reuse is identifying applicable content, which requires time and effort. Even though all possible reuse scenarios cannot likely be recognized from the very start of a documentation project, potential reuse cases may be identified and implemented at a later stage, which in turn may temporarily increase the writers' workload. However, investing in the effective reuse of content leads to cost savings in updating and reviewing content (ibid.). Although reuse offers many benefits, the reuse strategy cannot simply be "reuse everything". Forcing reuse may lead to trying to write content that is useful to all users, which likely results in too general content. As Stevens (2018) points out, the likely outcome of trying to please all users is that none of them is pleased.

Similarly, Schengili-Roberts (2018) notes that reuse is not automatically good; its value depends on how it is done. As an example, he points out that using the conref mechanism without a clear strategy is prone to lead to a tangled web of conreffed content without knowledge of where the original topic, phrase or some other type of reused content actually resides (ibid.). This could also result in broken links, and the user reading the output may think they are missing essential information when faulty links are replaced by whitespace. Even though content can be reused ad hoc by referencing content in its given location by assigning an @id attribute to an element and linking that element whenever necessary, unorganized reuse can result in difficulties with managing content, and writers may accidentally update elements without even noticing. To avoid this, the documentation team must establish a robust reuse strategy and have clear roles for who is managing that strategy, without forgetting the fact that they must communicate with each other about reusing content. As Self (2011, 180) states, reusing content is easier for writers if the reused elements are carefully organized in predictable locations. This is why it is recommended to appoint specific topics or DITA maps or both to function only as containers for frequently reused content, such as common phrases, variables and steps. Moreover, if an organization with multiple documentation teams uses DITA, it would be ideal for the teams to reuse their content according to similar rules because it would also facilitate cross-team content reuse.

The DITA standard offers features that can provide multiple benefits for writers and end users of documentation when they are followed. However, they must be consciously followed as DITA does not force any specific ways of working; organizations themselves decide whether they reuse content or which information types they use for certain types of content. Without having common guidelines and following them, writers from different teams – or even a single team – within an organization might opt for differing strategies and conventions, which in turn poses several challenges. Even if the differing strategies produced technically valid DITA, they impair the consistency of the content as well as reusing, minimalism and accurate semantic tagging, all of which serve the needs and expectations of the users of the documentation. As Kimber (2012, 2) states, the number of useful ways to apply DITA to documents is infinite, as is the number of possible useful writing practices that support DITA. As a solution, Kimber (ibid.) proposes assessing which practice or practices the end users need so that the optimal way to support the users' requirements could be determined, even if that means not using DITA.

The architecture and design of DITA underline that it is meant for task-centered documentation – the objective is to emphasize doing over knowing, which according to Stevens (2018) means prioritizing tasks over other information types, mainly concepts and references. However, the content whose information type is something else than giving instructions to complete procedures should not be forsaken either, as its function is to support the real-life tasks of the user. Stevens (ibid.) advises to ensure that concepts and references are not "orphaned", which means that if a concept's relation to a task cannot be established, it should probably not be included in the documentation at all. To map whether a topic should be included in the document may require researching the user and their needs. Of course, certain types of content such as legal information cannot be omitted even if they were not connected to any task in particular. In terms of technical documentation, it could be argued that behind every topic regardless of its information type is a real-life task, and naturally this should be reflected in the ratio of tasks versus other topic types in the documentation set.

In the end, even though DITA can be used in numerous ways, it might not be the ideal option for every organization or documentation team, such as those with a small documentation set without major reuse or localization needs, or those where subject-matter experts of the products are the ones who produce the documentation. However, if an organization wants to implement DITA, the architecture supports incremental adoption: the initial investment can be of a small scale and thus quick to implement, and it can be increased by implementing more capabilities

as the content strategy evolves and expands (Priestley and Swope 2008, 3). This applies to all users of DITA, and organizations can implement DITA more fully as their needs increase in sophistication and complexity (ibid., 18). Comprehending and following the principles of topic-based authoring, DITA and minimalism as well as the various mechanisms of the architecture enables writers to produce more effective and user-friendly information for the end users.

3 HEURISTIC EVALUATION

In this thesis, the research problem is to assess the DITA-authored customer documentation of four documentation teams in a business unit of the target company, especially in terms of how DITA has been applied in order to provide data for enhancing the documentation. The chosen research method is conducting a heuristic evaluation, which includes formulating a heuristics list for DITA. In the following subchapters, I first explore heuristic evaluation as a method and then introduce the DITA heuristics.

3.1 Heuristic evaluation as a method

Heuristic evaluation is a practical usability method developed by Jakob Nielsen for finding (and eventually fixing) problems in products, user interfaces and user manuals by evaluating their usability and other qualities based on a pre-defined list of established guidelines called *heuristics* (Korvenranta 2005, 111–113). The number of the applied heuristics depends on what is being evaluated and why. The method is fast and cost-effective, in addition to which it can be used at any phase of the product life cycle (ibid.). The flexibility of the method is implicated by Nielsen (1994b, 17), who describes heuristic evaluation as a "discount usability engineering technique" as opposed to the traditional usability research methods that involve complex tools or real users or both. However, Nielsen (ibid.) advocates for the method by remarking that the more sophisticated the usability testing method, the more expensive and thus unlikely to be used it is. Usability testing, especially when done with end users and by using complex equipment or software, can be expensive and time-consuming. As for heuristic evaluation, a simple research can be conducted by a single person with few prior preparations, and the results can be used immediately. Accordingly, Nielsen (ibid.) concludes that simpler methods such as heuristic evaluation are better posed to be put into practice during the product development, which is why this type of method should be viewed as a way of serving the user community.

A heuristic evaluation is conducted by a number of evaluators who independently and systematically view the product and list the benefits and drawbacks of it, preferably according to certain rules (heuristics) instead of intuition (Nielsen and Molich 1990, 249). The set of rules can be a pre-existing one or it can be developed specifically for the data in question (Nielsen 1994a). On the other hand, Korvenranta (2005, 122–123) argues that designing case-specific heuristics provides the greatest value in a heuristic evaluation, and the created heuristics can be

reused in an iterative process. The recommended number of evaluators is at least three but not more than five, because research has shown that three to five evaluators tend to find most of the usability issues whereas more evaluators would bring little extra value to the research (Nielsen 1994b, 155–157). Different evaluators will also find different types of problems. Depending on the circumstances, the evaluators can be experts of the domain or they can be nonexperts with little or no knowledge of the subject matter of the evaluation. However, the evaluators must have some experience of the heuristics so that they can apply them correctly (ibid., 20). In principle, the evaluators are free to decide how they want to proceed with the evaluation, but Nielsen (1994a) recommends processing the data at least twice so that the evaluators can first gain a general feel of the data and then focus on more scrutinized elements within the larger whole.

How the evaluation results are processed depends on the evaluation: they can be recorded as written reports, or the evaluators can talk their comments aloud as they proceed (Nielsen 1994b, 157). According to Nielsen (ibid.), the advantage of written reports is that they can be formally organized, but that requires an additional effort from the evaluators and a possible external observer to compile the reports. The report states the usability problem and the heuristic which the problem violates. Optionally, further notes such as the severity of the violation and a possible solution to the usability problem may be recorded as well. After the evaluators have conducted their individual research, the group convenes to discuss and compare their findings. Nielsen highlights the importance of this process because it ensures that each evaluator's report is independent and unbiased (ibid.).

The greatest advantage of the heuristic evaluation method is simultaneously its greatest disadvantage: it does not involve the end users of the product. Although users could provide new, valuable feedback and insight into the usability problems of a product, including them in a usability research would naturally require more resources. Heuristic evaluation instead is a flexible method that can be customized for any application and that does not require costly tools or training. However, in this thesis' case, the end users would not be able provide comments of the customer documentation since what and where they see is completely different from the DITA XML source format because they access the documentation either as PDFs or online web help. Additionally, the users' expertise lies within fields other than technical communication. Yet, conducting a heuristic evaluation in this type of a study could also provide information

about the underlying reasons for any possible usability problems that the end users may face, which makes them a fundamental factor in this research.

In the technical communication field, various heuristics have been published for different purposes. One of the most recent examples is the revised minimalism heuristics list by Virtaluoto, Suojanen and Isohella (2021) whose list is based on the original minimalism heuristics by Carroll and van der Meij in 1995. The revised heuristics are supplemented with the best practices of technical communication in the context of modern-day technical documentation and typical user instructions (Virtaluoto et al. 2021, 21). The minimalism heuristics were revised partly because at the time when Carroll created minimalism in early 1990s, software was still emerging, and the needs of the users were significantly different when compared to the today's digital natives. Virtaluoto et al. (ibid.) view minimalism as one example of a user-centered approach to good technical documentation, and the heuristics could serve as a practical, low-cost tool for practitioners.

Thanks to its flexibility, the heuristic evaluation method can be applied to a myriad of applications, such as games (see Pinelle, Wong and Stach 2008) and translations (see Suojanen and Tuominen 2015). In their MA thesis, Rautava (2018) created heuristics for evaluating user documentation in mobile applications. These applications of heuristic evaluation inspired me to test if the method could also be applied to DITA-authored source documents. In this thesis, heuristic evaluation is used as a method for gauging the extent to which the documentation follows the established general principles of writing content in DITA as well as the internal guidelines within the business unit. Heuristic evaluation has not yet been applied to XML source documents, which is why I compiled the DITA heuristics based on my theoretical and practical experience of authoring in DITA, the internal technical documentation guidelines and discussions with a senior documentation specialist of the business unit as well as the literature discussed in chapter 2. Another motivation for the DITA heuristics list is that according to Korvenranta (2005, 122–123), designing a customized heuristics list is the key for gaining the greatest value from a heuristic evaluation.

In this study, the findings of the evaluation are methodologically reported in an electronic form with notes and descriptions of the violations as well as suggestions for how the found issues can be attended at the business unit. Severity estimation is not a part of this research because the priority of the issues will be assessed during the on-going product documentation cycles. My task is to record and categorize the problems, collect meaningful data of them and propose

explanations as to why these issues have occurred and how they could be adjusted so that the customer documentation would better serve the end users. However, I am the only evaluator, which means that statistically I will not be able to discover all the possible issues within the data – according to Nielsen (1994a), a single evaluator finds approximately 35% of the usability problems, which I must acknowledge when I discuss my findings.

3.2 Heuristics for DITA

I compiled the 19 DITA heuristics for analyzing DITA topics based on literature on technical communication, the internal DITA guidelines of the business unit as well as my theoretical and practical knowledge of technical and structured writing. The heuristics list was created in an iterative process with a senior documentation specialist of the business unit who also reviewed the heuristics. My traineeship at the business unit also aided me in this process by providing me with an overview of the analyzed components' documentation. Therefore, the heuristics represent the real-life DITA usage of the writers at the business unit, and I presume that they will enable me to identify issues with the DITA-authored customer documentation.

This thesis examines two of the four main DITA topic types, concepts and tasks, and this emphasis is reflected in the heuristics. Heuristic 7.1 is only applicable for task topics. The third main topic type, reference, was omitted because it is not commonly used in the documentation of the analyzed components, and issues related to concepts and tasks are considered to weigh more from the user experience viewpoint. Nevertheless, the heuristics can be applied to and further customized for reference topics and any other information types as well. Furthermore, the heuristics are limited to address issues related to DITA topics only – they are not applicable to DITA maps. However, as they will be used internally in the business unit in the future, they will be extended so that they can be used to evaluate entire documents and documentation sets.

The DITA heuristics shown in Table 2 are designed to be used by writers. The purpose of the heuristics is to assess whether the main principles of DITA, minimalism and structured writing have been followed in DITA-authored documentation. Due to the scope of this study, the heuristics are not concerned with issues of grammar, real-life technical accuracy and style of the documentation. The heuristics are divided into eight categories that represent DITA's most significant features and principles in the context of the components' documentation.

Table 2. The DITA heuristics.

NUMBER	HEURISTIC
1	**INFORMATION TYPING**
1.1	Does the document type definition match the information type of the topic?
2	**TITLES**
2.1	Does the title have more than one word?
2.2	Does the title give a specific enough overview of the topic content (concepts) or state the user goal (tasks)?
2.3	Does the title match the content of the topic body?
2.4	Does the title match the guideline for the used document type definition?
3	**APPROPRIATE ELEMENT USAGE**
3.1	Have block elements been used correctly?
3.2	Have the correct semantic inline elements been used?
3.3	Has all taggable content been tagged with semantic inline elements?
4	**SEPARATION OF CONTENT AND FORM**
4.1	Is the topic free of highlighting elements for editorial purposes?
4.2	Is the topic free of empty elements?
5	**REUSE**
5.1	Have common variables been used for respective content?
5.2	Has textual content suitable to be a variable been used as one?
5.3	Has reusable content been reused?
5.4	Is the topic free of transitional information?
6	**TOPIC-BASED WRITING**
6.1	Does the topic have a single, identifiable purpose?
6.2	Is the topic self-contained?
7	**MINIMALISM**
7.1	Is the topic free of self-evident step or task results?
7.2	Is the topic free of repetitive or unnecessary descriptive information?
8	**LINKS AND REFERENCES**
8.1	Are references within the topic links instead of plain text?

The heuristics reflect the principles of minimalism and authoring content in DITA from a user-centered perspective. Moreover, they consider the business unit's current and future needs and the DITA practices that the writers of the unit are recommended to follow. Since I have been working as a trainee at the unit, I have gained a general understanding of its customer documentation and its characteristics. I have also considered the discussions I have had with other writers about their experiences and views on writing customer documentation.

The heuristics are formatted as detailed yes–no questions so that they could be used as a basis for a common tool by writers in the business unit. Consequently, the evaluation results can also be categorized as either positive ("yes") and negative ("no"). All heuristics will not be applicable to each topic, in which case the answer is non-applicable ("N/A"). The checklist-like structure with a limited number of possible answers enables reviewing the topics uniformly and compiling statistics of the results. When possible, the heuristics are listed in their order of presentation in a topic to make them easy to follow. Furthermore, the heuristics address re-occurring and probable issues on a detailed level which reflects the writers' work in real life. Next, all the heuristic categories are introduced based on the discussion in chapter 2.

1. Information typing. Heuristic 1.1 ("Does the document type definition match the information type of the topic?") refers to the categorization of topics according to the type of information that they contain. The information type of a topic is undoubtedly the first choice a writers makes when they create a new topic. Writers should use the appropriate information type because DITA is designed around a topic-based information architecture (OASIS 2015, 27). This disciplined approach to writing emphasizes modularity and reuse of concise units of information (ibid.). Choosing the correct topic type is crucial for the sake of modularity, reusability and minimalism. Stevens (2018) raises another benefit of using the correct information type: the DITA schema enforces structural consistency, and thanks to that end users can anticipate where information is located within a topic. Therefore, following the DITA structure provides predictability to users. This means that even if writers found ways to rearrange elements as they like, it would not be advisable because it could confuse the users.

2. Titles. The <title> element is the only mandatory element in a topic – therefore how it is formatted is important. They are used both as a core navigation aid and for information labeling (Self 2011, 131). Moreover, the title is likely the first line of text that the user reads, and the informativeness and accuracy of the title may also be the decisive factor in whether they will continue reading or not, which is the justification for heuristic 2.2 ("Does the title give a specific

enough overview of the topic content (concepts) or state the user goal (tasks)?"). This heuristic is slightly different for the two topic types. It also ensures that the title is not ambiguous or that it does not cover multiple ideas. Vague one-word titles are not user-friendly, and they are not detailed enough to produce accurate search engine results either, which is why heuristic 2.1 ("Does the title have more than one word?") exists. The title "Overview", for example, does not tell the user anything about what the topic content is about, and thus it cannot stand alone (ibid., 132). In addition, if the documentation is located in an online help center, the search results for a query with "Overview" would not produce very accurate results for the busy user – a single document set could include multiple topics that are an overview of something.

Heuristic 2.4 ("Does the title match the guideline for the used document type definition?") refers to the internal guideline that specifies how the title should be formatted for each topic type. For concepts, the title should be a noun phrase of more than one word that outlines the topic content unambiguously and in sufficient detail. It should start with either a noun or an adjective, not a verb. For tasks, the instruction is to cover the real-life user goal of the procedure and to begin the title with the gerund (*-ing*) form of the main action verb. Consistently formatted titles help users recognize whether a topic is an instruction or a description. The answer to this heuristic depends on which topic type, not information type, is used (for example, if a concept topic describes a procedure and its title is formatted with a verb in the gerund form, the answer is "no", because the task DTD has not been used).

Heuristic 2.3 ("Does the title match the content of the topic body?") was not originally included in the list, but it was added in the early stages of the analysis when it was noticed that some topic titles were misleading in the sense that they referred to something that was not clearly covered in the topic content itself. It overlaps with heuristic 2.2 partially but is different in the sense that this heuristic is for ensuring whether the documented content in fact covers the end-goal or concept that is stated in the title or not.

3. Appropriate element usage. Elements are the building blocks of XML. *Block elements* (e.g. <p>, <steps>, <table>) are paragraph-like elements that are typically displayed in the output with space above and below the content, whereas *inline elements* (e.g. <uicontrol>, <filepath>, <codeph>) are applied to words or phrases within a block element typically without forcing empty space around them (Self 2011, 64). The number of available elements is considered to be a challenging aspect of writing DITA (Bellamy et al. 2011). In 2022, DITA 1.3 has a selection 611 elements in total (OxygenXML 2022), all of which have a specific meaning or

purpose, and if the organization has specialized its DITA, the number could be even higher. Still, writers are not expected or even recommended to use all 611 elements. In addition to the 182 base DITA elements[4], the documentation teams of the analyzed four components use 39 other semantic elements from the business field of the business unit. These 39 elements are from the software, user interface and programming fields[5] since the business unit produces software products that are operated either via a graphical user interface or a command-line interface. Thus, the number of allowed elements in the analyzed documentation is 221.

In both DITA and most XML languages in general, elements carry semantic meaning which conveys metadata about the type of information that specific element contains. Each information type is associated with a document type definition (DTD) that has a varying set of available semantic elements – elements that are not specified in a topic's DTD cannot be used in that topic. However, DITA cannot guarantee using tags so that they correspond to what the meaning of the content is, nor can it confirm that tags are applied consistently – DITA enforces the validity of the content by ensuring the appropriate order (syntax) of the elements. Thus, writers should supplement the content with the proper elements according to their intended purpose. Heuristics 3.1 ("Have block elements been used correctly?") and 3.2 ("Have the correct semantic inline elements been used?") are interested in whether block and inline elements are used, while heuristic 3.3 ("Has all taggable content been tagged with semantic inline elements?") is for checking that tagging relevant content has not been dismissed. To summarize, semantic markup plays an important role in the findability, categorization and output processing of the customer documentation.

4. Separation of content and form. Structured writing is characterized by the objective of separating content from form in order to ensure the possibility to effectively single-source and reuse content as well as to guarantee a consistent look and feel of the output formats. Writers should not endeavor to affect the end-format by using typographic elements because this leads to ignoring the proper semantic element and adding unnecessary emphasis. In the DITA standard, these elements are referred to as highlighting elements, and the instruction is to never use them when a semantically specific element is available (OASIS 2015, 293). On this note, most of the semantic elements, like <uicontrol> for user interface elements, are formatted

[4] Base elements (e.g. <topic>, <body>, <p>, <ph>, <topicref>) are the core elements that enable specializations.

[5] These element categories are included in the technical-content domains that are categorized by OASIS according to the type of the content to which they are associated.

differently in the authoring tool GUI that differs from the body text, most often **bold** or `monospace`, the latter of which mainly applies to elements for program code. However, that does not mean that is how they are formatted in the final output. The appearance of each tag can be configured to be delivery format-specific, meaning that how the output looks may differ between the deliverable formats. This is why Self (2011, 125) notes that if one endeavors to deliberately code DITA topics in such a way as to achieve a particular output effect, they are undoubtedly taking the wrong path – DITA topics must be delivery-agnostic and free of as much context (usage, presentation or sequence of the information) as possible. Self (ibid., 229) continues to argue that rather than thinking that a word needs to be displayed in bold, writers need to think about what it is that makes it necessary to distinguish that word from the others.

Heuristic 4.1 ("Is the topic free of highlighting elements for editorial purposes?") is for checking whether highlighting elements have been used in an attempt to affect the output's appearance. This subject is significant to the analyzed documentation because according to the senior documentation specialist of the business unit, the possibility of having <b> fail the output processing was considered as an option to prevent the use of highlighting elements. However, this was not implemented since a single project can have thousands of legacy modules, and reserving resources for making purely editorial corrections to the existing modules instead of creating new content does not have a valid business case. In turn, heuristic 4.2 ("Is the topic free of empty elements?") is for recording empty elements because some of them might cause blank lines (whitespace) in the output.

5. Reuse. Reusing content decreases the time, effort and resources needed for documentation. Reusing is enabled through single-sourcing, which means that when the source is edited, all referenced instances of that source are updated automatically. In both theory and practice, any DITA fragment (a single word, a phrase, a paragraph, or a whole topic or DITA map) is eligible for reuse. However, all content should not be reused. Reuse is often done to ensure easy updating via variables, which are keyreferenced entities that are used by replacing their value when the information is published. In this thesis, the common variables mentioned in heuristic 5.1 ("Have common variables been used for respective content?") are the proper names of the target company and the documented products, applications and portals. Heuristic 5.2 ("Has textual content suitable to be a variable been used as one?") is for identifying potential new variables that have been hardcoded, in other words plain text that is not defined as a key.

Content is also reused to avoid rewriting information and to ensure consistency. Separating information into discrete topics by information type helps writers identify common or reusable topics or pieces of content in topics (Bellamy et al. 2011, ch. 1). For example, similar task topics may share common steps. If the exact same information can be repeated in numerous topics, it makes little sense to write it again each time. In addition, maintaining and updating content is easier and quicker when some of it is reused. The best practice would be to write content once according to the relevant guidelines and to reuse and update it, which is why heuristic 5.3 ("Has reusable content been reused?") is for checking whether reusable content has been reused, and if not, for identifying potential reuse cases.

Heuristic 5.4 ("Is the topic free of transitional information?") concerns phrases or other linguistic devices that refer to information in previous or following topics, which decreases the reuse potential of DITA content – topic-based information should not require external context in order to be understandable by the expert target audience. Well-designed topics can be reused in many contexts as long as unnecessary transitional text is carefully avoided (OASIS 2015, 27). The business unit's guidelines allow using transitional phrases reasonably within a single topic when referring to an object in the immediate presence of text, such as a table or an image right above or below a text paragraph. More often than not, transitional text is not even necessary for understandability, as DITA has its own mechanisms for maintaining cohesion and managing transition. These include meaningful titles, automatic labels (e.g. "Result:" or "Outcome:" for a <stepresult> element) and structural choices, such as writing modular topics and chunking them (collecting several topics to appear as one HTML or PDF page) with the @chunk attribute in the DITA map. Superfluous transitional information can also be avoided by adhering to the minimalist writing principles (Self 2011, 126).

6. Topic-based writing. Similarly to elements, topics are the building blocks of DITA documents. A topic is a standalone chunk of information that is typically limited to one idea, and each topics should be stored in its own file (ibid., 18). As a general rule, topics should be self-contained, which means that they are independent units of information that the users can exploit to accomplish a goal, whether that goal is learning about something (concepts) or completing a procedure (tasks). Topics should also be written according to the principles of information typing (heuristic category 1). As Self (2011, 19) states, the smaller the topic unit, the greater the opportunity for reuse. In addition, self-contained and independent topics are

more reusable than topics that are dependent on the content of other topics, or their location or hierarchy in the DITA map, for example.

Topics should be as short as possible but long enough to be independent of other topics, which means that they must make sense on their own in any context. For example, task topics should cover the entire procedure that the user must complete in order to reach their goal. Concept topics instead should have value-adding, meaningful content that does not repeat itself. Topics that do not seem to convey any information or that are missing essential information for understanding the subject of the topic contradict heuristic 6.2 ("Is the topic self-contained?"). On the other hand, topics that encompass various user-goals or that mix multiple information types by containing multiple concepts or ideas or by using other workarounds conflict heuristic 6.1 ("Does the topic have a single, identifiable purpose?").

7. Minimalism. Instead of taking a system–component-oriented approach, writers should follow the minimalist principle of emphasizing the user and their goals – users want to find the piece of information they need as quickly and effortlessly as possible without having to spend time on reading nonessential information so that they can start and complete their task without delay. Thus, unnecessary or repetitive text hinders the user in their work. Elements are the building blocks of DITA, but there is a risk of using an element simply because it can be used and also seems useful, even in cases where the topic does not require that type of information. Oftentimes, a well-written element can make another one redundant – a common example of this would be the <context> element that provides background information for a task. In some cases, a well-formulated topic title in conjunction with a well-written <shortdesc> can also provide sufficient information, thus obviating <context>. Still, a linear way of thinking and writing might result in repeating the title or <shortdesc> content in <context>, which is both against the internal guideline and a violation of heuristic 7.2 ("Is the topic free of repetitive or unnecessary descriptive information?"). Example 8 shows this type of error:

(8) <task><title>Troubleshooting</title>
<context>This section describes troubleshooting.</context>
</task>

Of course, the user will know the goal of the task topic in example 8 without the information included in <context>. In fact, they might be irritated of having had to read the exact same information twice. Therefore, the <context> element should either be omitted or rephrased and further expanded if the user should have more information about the background of the task.

The other minimalism heuristic (7.1 “Is the topic free of self-evident step or task results?”) applies to task topics only because the need to avoid self-evident content can be specifically striking in procedures with steps. For instance, the <step> element can contain a single <stepresult>, but some step results are obvious and therefore not worth stating. The following is an example of a self-evident step result:

```
(9) <steps>
      <step>
       <cmd>Click<uicontrol>Install</uicontrol>.</cmd>
        <stepresult>The <uicontrol>Install</uicontrol> dialog box opens.</stepresult>
      </step>
     </steps>
```

Any complex or possible unexpected outcomes that require troubleshooting are more interesting to the user than the ones that the user can figure out on their own, and naturally any unexpected or potentially harmful results that must be remedied should be stated.

8. References and links. References are for reusing content, while links guide the user to additional information. However, the user does not want to see links to content that is irrelevant in terms of their current task or question, and they certainly do not want to feel as if they are lacking something if a link in the middle of a clause does not work or is missing – the fallback mechanism in oXygen is that broken inline links are omitted from the output, leaving whitespace in the body text. Also, embedded links create a dependency upon the linked topics as the topic can only be understood if the target topic of each link is delivered alongside it (OASIS 2015, 24). Therefore, a specific element, <related-links> should be reserved for links in topics, since it gathers the links safely at the end of the topic in a bullet list, and if a link were broken, its bullet and content would simply be omitted. However, although <related-links> has been discussed internally, it has not been included in the guidelines, which is why its usage in the analyzed DITA topics is excluded from the heuristics. Heuristic 8.1 (“Are references within the topic links instead of plain text?”) is for ensuring that references to content are made by using the DITA mechanisms instead of plain text in order to ensure proper reuse.

The next chapter introduces the sample data of 80 DITA topics which were evaluated according to the 19 DITA heuristics. The heuristics’ suitability is discussed in the conclusions chapter.

4 DATA

This chapter outlines the research data as well as its selection and collection processes.

4.1 Customer documentation in DITA

The data for this study was originally produced by small teams of writers from the four software components in the business unit. The four components that were selected for this study have their own DITA-authored customer documentation, which the users can view in both HTML and PDF format in a customer documentation portal or as software-integrated HTML. The target audience of the documentation is expert users. The four components, A, B, C and D, whose files are analyzed are involved in a transformation project in the business unit, which is the reason for their selection.

The analyzed materials in this thesis are DITA-authored XML files. Each file represents a single DITA topic of varying lengths. The sample consists of concept and task topics because they are the most common topic types in the customer documentation. Topics were chosen for this study instead a whole collection of topics organized in a DITA map because the business unit wants to study whether its documentation consists of self-contained modules for reuse purposes and other benefits of DITA which were covered in chapter 2. Choosing topics as the research data also enabled me to analyze and evaluate certain aspects of the documentation without any context or background information of the components. Another reason for this restriction is that the customer documentation of complex components typically consists of multiple DITA maps or one massive DITA map for a documentation set, and therefore the size of the data would exceed the scope of a master's thesis. As the research problem is to evaluate how DITA XML has been used, aspects relating to the quality, grammatical correctness or real-life technical accurateness of the documented content were excluded from the research scope.

Since the aim of the study is to conduct a heuristic evaluation of the business unit's customer documentation, the first criterion for the data was that it must have been delivered to the end users. Otherwise, I could inadvertently evaluate severely outdated or unfinished work, which would distort the findings. This meant that I had to pinpoint a certain timeframe during which all four components had been released. I chose to narrow the timeframe to the second quarter of 2021 because it would also fulfill the second criterion: selecting a sample that has not been

affected by the on-going transformation project. The technical writing teams may already have started implementing some of the guidelines that have been introduced internally – this restriction is important because the aim of this study is to gain an understanding of the starting point of the development. Since the project had not yet started in the second quarter of 2021, the files from that time period have not been updated by anyone who is aware of the project.

4.2 Sample size and collection

The documentation teams of the four components of the business unit manage their content via Git, a popular version control system for environments in which multiple people work on the same projects simultaneously. Git is a distributed version control system, which means that each user works on their local copy of the whole history of the project and synchronizes their work whenever necessary while Git tracks the project history in a database called *repository*. By recording the work of each contributor, Git allows the travelling back in time to snapshots of the repository with the concurrent documentation. It would also enable me to retrieve any version of a repository from any specific period of time, including the second quarter of 2021, and download a local copy of the customer documentation at that moment to collect the sample for the heuristic evaluation.

Next, I needed to determine a purposeful sample size for my research. According to Eskola and Suoranta (1998, 61–62), qualitative research does not enforce a particular sample size, nor does the size strictly determine whether the study will be successful or not. To my knowledge, DITA-authored source files of real-life customer documentation have not yet been researched, which means that I could not base my sample size on previous research. Instead, I had to use my own judgement on the selection of the number of files for the analysis. The number must, however, be large enough to provide a reasonably sized pool of material to analyze without exceeding the scope of the research. To determine a suitable sample size for this thesis, I first counted how many concepts and tasks the documentation set of each of the four components had within the researched timeframe. With that information, I could estimate a manageable sample size for the heuristic evaluation.

To obtain that statistic, I searched the two evaluated information types, namely the concept and task topics, in the local copies of the four respective repositories. Table 3 reports the total number of concept and task topics per each component and their percentage of the whole:

Table 3. Total file count and distribution of the components.

Release in second quarter of 2021					
Component	**Number of files per type**		**Number of topics per component**	**Percentage of total**	**Number of files for analysis (50% concepts, 50% tasks)**
	concept	**task**			
A	1226	161	1387	41	30
B	539	303	842	25	20
C	773	124	897	26	20
D	101	169	270	8	10
total	2639	757	3396	100	80

Although the total number of topics in the four repositories is approximately 3400, it does not truthfully reflect the number of actual topics in the customer documentation; the numbers likely contain obsolete files which have not yet been deleted from the repository, and simultaneously it is probable that some duplicate content also exists.

Since I would be the only evaluator, the number of evaluated topics must be manageable for one person so that the results could be recorded on a sufficiently detailed level to enable using them in the transformation project and any other relevant projects in the future. But how many topics should I evaluate? As Eskola and Suoranta (1998, 61) explain, qualitative studies are more interested in the how well the data represents the researched phenomenon than how great the quantity of the data is. The collected data should be relevant to the study, in which case the size of the sample might also be relatively small (ibid.). In addition, in qualitative research, individual findings typically weigh more than frequencies or other types of numerical data. Eskola and Suoranta's (ibid., 62–63) minimalism-spirited advice is to choose a sample size as small as possible to answer the research questions. To be able to determine this sample size necessitates prior knowledge of the researched data and what can be expected to discover from it. I estimated that I could evaluate dozens but not hundreds of topics. In fact, analyzing hundreds of topics could prove to be rather meaningless – this argument is related to the idea of *saturation of data*.

According to Eskola and Suoranta (ibid.), data becomes saturated when analyzing new instances of it halts to yield novel information about the data. As each valid task or concept topic would unequivocally follow the same rules that are defined in the corresponding DTD, it could be safely presumed that the studied files would yield similar answers to a certain degree. Of course, the saturation point is different for every research, and it cannot necessarily be calculated beforehand. In that case, Eskola and Suoranta (ibid.) propose determining the saturation point as the research progresses. I decided to follow this advice and estimated

together with the business unit's senior documentation specialist that the saturation point for four components could be less than hundred but more than fifty topics. Because topics can be as short as having the root element and a title, it was decided that at least 10 files (five tasks and five concepts) from each component should be chosen to ensure that there would be enough material from each component for the evaluation. It was decided that I should collect a sample of 80 topics while considering the size differences of the four components' repositories. The share of the respective components' topics and the number of files for analysis are reported in Table 3. If the sample would not reach the saturation point, I could collect additional topics for the evaluation at a later point in the research. In the end, there was no need to increase the sample size.

Finally, I needed to collect the sample. The chosen sampling method was simple random sampling because it would yield a representative sample of the entire population, in addition to which it would minimize the risk of researcher bias. Each file would have an equal probability of being selected to the sample. I used an online randomizer tool to recreate the alphabetized list of the components' concept and task topics, and then used a random sampling tool to generate a random set of numbers according to Table 3 from the number range of the randomized file list. When I started the random sampling, I noticed that some of the filtered topics did not seem to be referenced in any DITA map, which means that in the second quarter of 2021, those files were not a part of the customer documentation set that was released to the users. To check that any these files would be excluded from the sample, I used an automated function of oXygen that did not reveal the topics' context to me, which was a requirement for the heuristic evaluation.

In the next chapter, the 80 DITA topics are evaluated according to the DITA heuristics.

5 HEURISTIC EVALUATION OF DITA TOPICS

This chapter presents the heuristic evaluation of the four components' DITA-authored customer documentation. In the analysis, I assess both the individual components as well as the differences and similarities between them by focusing on the main themes of the findings. Since the data comprises 80 topics, I will use case examples of various heuristic violations to illustrate which types of issues were recognized in the sample.

I evaluated each topic in isolation without the DITA maps in which they are contained, which meant that I could not see where the topic is located in the topic collection. Accessing this information would have distorted the analysis because one of the evaluated themes was assessing if the modular, topic-based design of DITA had been implemented correctly – DITA-authored content is ideally free of form and context and should, therefore, be usable and understandable on its own as if any topic were the first page of a document (PDF) or a collection of topics (HTML). Table 4 exemplifies how the 80 sample topics were listed according to the component and topic type, and how the answers were color-coded for the analysis.

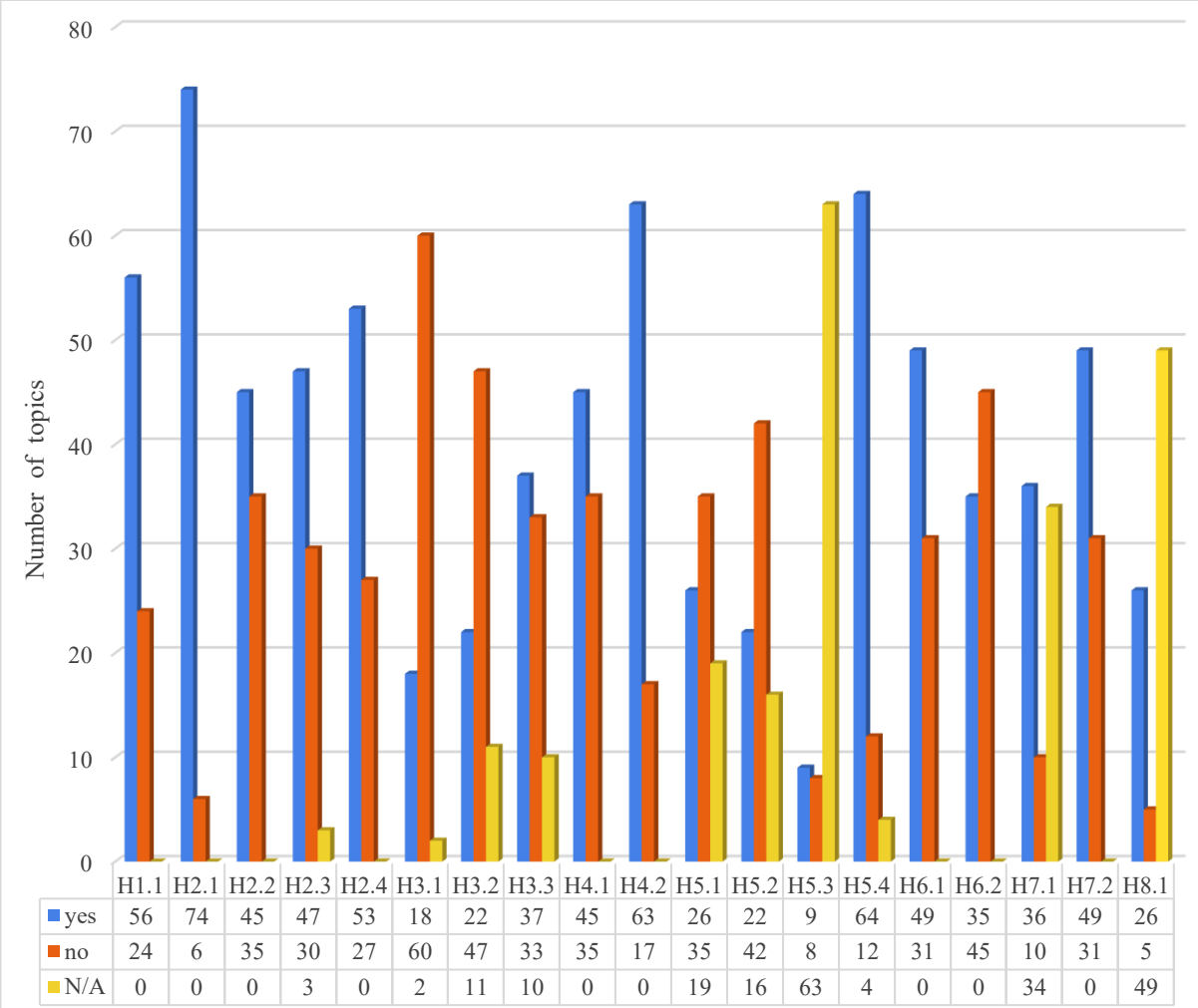
Table 4. Heuristic evaluation report example.

Topic file name	Component	Topic type	Heuristic 1.0	Heuristic 2.1	Heuristic 2.2	…	Heuristic 8.1
[file_1]	D	concept	yes	yes	no	…	N/A
[file_2]	D	concept	no	yes	yes	…	no
[file_8]	D	task	yes	yes	yes	…	N/A

The results of the heuristic evaluation are represented in Chart 1. The abbreviation *H* denotes the number of the heuristic in question. If the answer is "yes", the heuristic had been followed in the topic. If the answer is "no", there is an issue regarding the heuristic. The third answer category, "N/A" (non-applicable), is used when the heuristic cannot be applied to the topics (for example, if the topic does not have taggable content, heuristic 3.3 ("Has all taggable content been tagged with semantic inline elements?") cannot be applied) for some reason.

The examples of the analyzed data are formatted as XML code as it would be shown in oXygen's Text mode. Their language has not been edited, and all references to internal content have been anonymized and *italicized*. Some of the examples are shortened so that only the relevant issues are included. In other words, all examples do not represent whole DITA topics.

Chart 1. All results.



	H1.1	H2.1	H2.2	H2.3	H2.4	H3.1	H3.2	H3.3	H4.1	H4.2	H5.1	H5.2	H5.3	H5.4	H6.1	H6.2	H7.1	H7.2	H8.1
yes	56	74	45	47	53	18	22	37	45	63	26	22	9	64	49	35	36	49	26
no	24	6	35	30	27	60	47	33	35	17	35	42	8	12	31	45	10	31	5
N/A	0	0	0	3	0	2	11	10	0	0	19	16	63	4	0	0	34	0	49

This chapter is divided into eight subchapters according to the heuristic categories and the themes of the observations of the analysis. These discussions are followed by a brief evaluation of the suitability of the DITA heuristics in this study.

5.1 Information typing and modular writing

Matching the information type with the correct topic type is the first heuristic (1.1 “Does the document type definition match the information type of the topic?”) as well as the first decision when creating a new topic. Categorizing information according to its type is not always straightforward, especially with complex products where neither the concepts nor the tasks are simple. However, DITA attempts to enforce using the most appropriate topic type by assigning strict syntax rules for the types that are specialized from the base *topic* type. As Self (2011, 97) says, the task topic is the most *strongly typed* topic of the three original base information types

(concept, task, reference), meaning that the task DTD rules are the strictest. In brief, the rigidness of the information type makes it difficult to use a task topic for some other information type. Therefore, I anticipated that concepts are more likely to be used incorrectly because their DTD rules are more lenient than those of tasks. This assumption proved to be correct: only 5 out of 40 task topics should have been either concepts or references. On the contrary, altogether 19 concepts of the 40 conflicted heuristic 1.1 ("Does the document type definition match the information type of the topic?"). Of these 19 concepts, five should have been reference topics because they mostly consisted of tables and lists and did not include enough information to give an overview of the subject or answer the question "What is…?" (see Table 1).

The remaining 14 concepts were so-called "fake tasks", in other words concepts that should be tasks. The dilemma with using a concept instead of a task is that the task-specific semantic elements cannot be used in concepts because they are not defined in the concept topic DTD. This problem can be ignored by using less strongly typed elements to produce a similar-looking output as with task-specific ones, but this practice does not conform with DITA – content should be separated from form, and it should be information typed. If, however, a concept is formatted as if it were a task, the end-result is this type of a structure where a <li> ("list item") element nested within an <ol> ("ordered list") replaces multiple task-specific semantic elements:

(1) (Component A, file 51)

To deploy the [*practice*]:

Select a <ph keyref="*product_name_A*"/> environment from the Environment dropdown list. For instruction on how to set up <ph keyref="*product_name_A*"/> environment, see <xrefhref="*configuring_A_environment.dita*"/>.

Select one of the rows from the list of file types. For example, click on Processes to deploy a workflow. You see a list of files in your workspace (with L icon) and in the selected <ph keyref="*product_name_A*"/> environment (with R icon).

If the correct topic type, task, and its specific elements were used in example 1, both <li> elements of example 1 would be <step> elements, and the two commands ("Select…") would be <cmd> ("command") elements. The <cmd> element is specified to include only the action that the user must perform as a single sentence with the main action verb in the imperative form and nothing else. In example 1, both <li> elements include more content besides the command. In a task topic, the text "For instruction on how to…" from the first <li> would be included within an <info> or, if appropriate, a <prereq> element. If the second <li> were a <step>, it would include at least two other semantic elements: <stepxmp> for the example ("For example, click…") and <stepresult> for the result of the step ("You see a list of files…"). In addition to

losing the semantic element information for the more appropriate elements, any special formatting rules, such as labels or line spacing that the task-specific elements would have, are lost in the output process. This is because DITA separates content from form, and the appearance of each element is defined elsewhere than the source XML file. For instance, in the output formats of the documentation, the <stepresult> has a label that reads "Expected outcome" followed by an empty line between the label and the element content. The output for the <li> element instead does not include an informative label or line breaks. Therefore, using the incorrect topic type will affect both the information model and the appearance of the content.

Similarly to using an incorrect topic type, mixing information types in a single topic can confuse the user – if they search and find a topic that describes what something is, they will not likely expect it to contain procedural information because of the topic-based information structure of the documentation where step-by-step instructions and definitions are contained in separate topics. As an example of mixing information types, a task topic of component B (file 21) first describes the naming conventions of an object on a very detailed level in the <context> element (which is for providing background information for a task), after which the topic introduces a long, multi-row table with all the different naming possibilities for the object. This conceptual (<context>) and referential (<table>) information is followed by step-by-step instructions for configuring the object names, which means that the topic includes altogether three different information types. This practice does not conform to DITA, though, because information typing is also connected to the idea of modular writing where a topic should only cover a single idea or answer a single question (Self 2011, 19). Tasks are separated from extensive conceptual or reference information so that tasks remain short, retrievable, and reusable (Bellamy et al. 2011, ch. 2). If tasks are overloaded with too much conceptual or reference information, expert users become frustrated because they must wade through information that they might already understand (ibid.). Information typing may cause some topics to be extremely short. However, self-containing topics, especially tasks, may also contain some essential descriptive information to avoid having to point the user to another topic via an <xref> link if they could instead access that information in the immediate vicinity of the task or task step itself.

The sample contained another typical instance of incorrect modular writing, which is combining multiple procedures into a single topic by nesting step-by-step instructions with <section> elements in a concept topic. In component D, a concept that describes sharing and managing user profiles a UI contains altogether five <section> elements:

(2) (Component D, file 10)

```
<concept>
    <title>Profile sharing user scenarios</title>
      <conbody>
        <section>
          <title>First Scenario</title>
          <p>[conceptual information of scenario 1]</p>
        </section>
        <section>
          <title>Workflow to execute scenario 1</title>
          <ol>[6 <li> steps for executing scenario 1]</ol>
        </section>
        <section>
          <title>Managing contributors</title>
          <p>[conceptual information of contributors]</p>
        </section>
        <section>
          <title>Second scenario</title>
          <p>[conceptual information of scenario 2]</p>
        </section>
        <section>
          <title>Workflow to execute scenario 2</title>
          <ol>[6 <li> steps for executing scenario 2]</ol>
        </section>
      </conbody>
</concept>
```

The <section> element is meant for dividing topics into subsets of information that is directly related to the topic (Self 2011, 20). Still, simple topics should be preferred to sections because sections result in hierarchy and sequence in the document structure as they are embedded in the topic, in addition to which they tend to cause topics to be long as in example 2, and long topics are more difficult to reuse (ibid., 20–21). The procedures for the scenarios in example 2 are related to the same subject, but despite their close resemblance they should not be contained within the same topic as sections. Instead of creating long procedures in a single topic, the topic could be separated into several shorter task topics that could be assembled into a logical order that helps users to finish the entire procedure (Bellamy et al. 2011, ch. 2). Writing one procedure per task topic makes managing, organizing and reusing the topic easier, and it also serves the end users' needs since they typically look for a single procedure at a time and want to be able

to find it when they need it (Bellamy et al. 2011, ch. 2). In example 2, a user who needs the procedure for the second scenario must first read through the three sections before reaching the information that they need to accomplish their real-life goal. Therefore, the topic should be divided into four concepts (one for the title, two for the scenarios and one for the conceptual information of managing contributors) and two tasks for the two scenarios or workflows.

In other words, the findability of information is an essential part of customer documentation, as is the reuse potential of topics in a DITA-authoring environment. Using the incorrect information type as well as combining information types or multiple ideas in a single topic detract from the design of DITA and the principles of minimalism and task-orientation. A possible explanation for these types of solutions is the intention to create topics according to how they should be presented in the deliverable reading format. This, however, is not recommended by Self (2011, 19), who instead suggests joining closely-related topics by using the @chunk attribute in the document's DITA map so that they would be delivered under a single heading. For instance, the two procedures (scenarios) of example 2 could be chunked together under the concept topic, or if they must be performed together, they could be contained within a single task topic by using <substeps> and a different title. The @chunk attribute is not used by all documentation teams at the business unit, while the internal guideline recommends using chunking for concepts only. Based on the analysis, it seems that writers have used other methods to achieve the same effect – with overusing <section> being one of them.

5.2 Formatting titles

Similarly to information typing, the formatting of the topic title is an essential part of topic-based writing in DITA – they are used for both navigation and information labeling purposes and should, therefore, be meaningful, easy to read, accurate, concise and consistent (Self 2011, 131–132). Consistency also helps the users find the information they seek – when different topic types have their own, easily distinguishable titles, the user can instantly recognize what type of information they will find by reading that particular topic. The business unit's guideline instructs using noun phrases for concepts, while task titles should begin with the main action verb in the gerund form (*-ing*). Out of the 80 topics, 27 violated heuristic 2.4 ("Does the title match the guideline for the used document type definition?"). The number for task topics was five, two of which had the main action verb in the imperative form, and three were noun phrases. Slightly over 50% percent of all concepts were not noun phrases, and most of them were

formatted according to the task topic guideline. This is particularly confusing from the users' point of view: for example, if the user is looking for instructions for configuring parameters and they click a topic titled "Customizing [object] parameters" (title of a concept of component C, file 73), they will be disappointed to notice that the topic they just clicked and spent time searching for in fact contains conceptual information about the subject instead of instructions for reaching their goal. Unintentionally making the user waste time and effort can be avoided by following the business unit's guideline of how titles of the various information types should be formatted because this enhances the predictability and thus the findability of information from the users' viewpoint. When used correctly, titles are a crucial piece of the puzzle of finding information – accurate and consistent titles make content easy to find, and consequently save the users' time (Stevens 2018). This issue is intrinsically tied to proper information typing.

On the contrary, vague one-word titles such as "Overview" or "Limits" complicate the puzzle, as the user may need multiple search attempts to find what they are looking for. It was anticipated that the sample would include one-word titles, even though the internal style guide advises against them. Six concepts only (four from component A and two from component C) contradicted heuristic 2.1 ("Does the title have more than one word?"). Based on their vast experience, a senior documentation specialist of the unit commented that writers who use titles such as "Introduction" might be writers who author documents for the PDF format only, which represents the more traditional, book-like style of writing and assembling user guides instead of the HTML format. Thus, the writers may assume that the context is known to the reader, perhaps via the document title or an upper-level section title, but in a topic-based, structured writing environment, one-word titles make topics less findable and less self-contained.

Heuristic 2.4 also revealed how many topic titles start with the words "How to". The internal guideline strongly advises against using this construction in task titles. If task titles were formatted as "How to" questions, the users could not scan the title quickly, because its two first words are fundamentally redundant as they do not tell anything about the topic content. In a "How to" title, the third word would likely be the action verb, but this is contradictory with the first minimalism principle: choose an action-oriented approach that enables the users to get started on a real-life task without delay. The user will likely focus their eye to the first word of the sentence, whether that is done intentionally or subconsciously, which is why the very first word should be emphasized instead of the third one. The sample contained only one topic with this type of title, so it seems that this instruction is widely recognized.

According to the minimalist principles, technical documentation should avoid the so-called system-oriented approach where the functionalities of the product are emphasized with the cost of ignoring the real-life user goals. For instance, one of the task topic titles (component A, file 38) is "Submitting a Descriptor for testing", which is precisely what the task steps instruct the user to do. However, *submitting* something is not the user's end-goal – what they want to do is *test* the descriptor. This conflicts with heuristic 2.2 for tasks ("Does the title state the user goal?") as well as the second principle of minimalism: emphasize users' real-life tasks. The main action verb *submitting* of the task topic's title may be explained by the fact that the command of the last step in the task instructs the user to click the *Submit* button, and the name of this UI control has ended up in the title. But as Hackos (2012, 1) argues, even though the user does operate in the UI, clicking a button is never their real goal.

The most surprising results for the title heuristics were those of heuristic 2.3 ("Does the title match the content of the topic body?"), which was added to the heuristics list during the analysis. Nearly 40% of the topic titles were in conflict with the content of the respective topics. This is a major deficiency for many reasons. If a task topic's title states a user goal but how that goal is reached is not covered by the procedure, the user will likely be alarmed – if their goal is not covered by this topic, what would be the correct search terms for finding the correct topic in the documentation – or has such topic been included at all? Or if they follow the steps, will they accomplish their goal or not, and if not, why is that? In example 3, the title of the task topic suggests that the task covers the procedure for restoring a notification server, but the task body only includes instructions for verifying whether the restoration is successful.

(3) (Component D, file 5)

```
<title>Restoring notification server on <ph keyref="product-short-name"/></title>
<taskbody>
  <steps>
    <step>
      <cmd>Verify that the notification server configuration maps are restored.</cmd>
      <info>
        <codeblock>[program code]</codeblock>
      </info>
      <stepresult>The following configuration maps must be listed.
        <codeblock>[program code]</codeblock></stepresult>
    </step>
```

A more appropriate title for this task could be "Verifying the restoration of the notification server". After reading this task, the user might be left with the question of whether they now

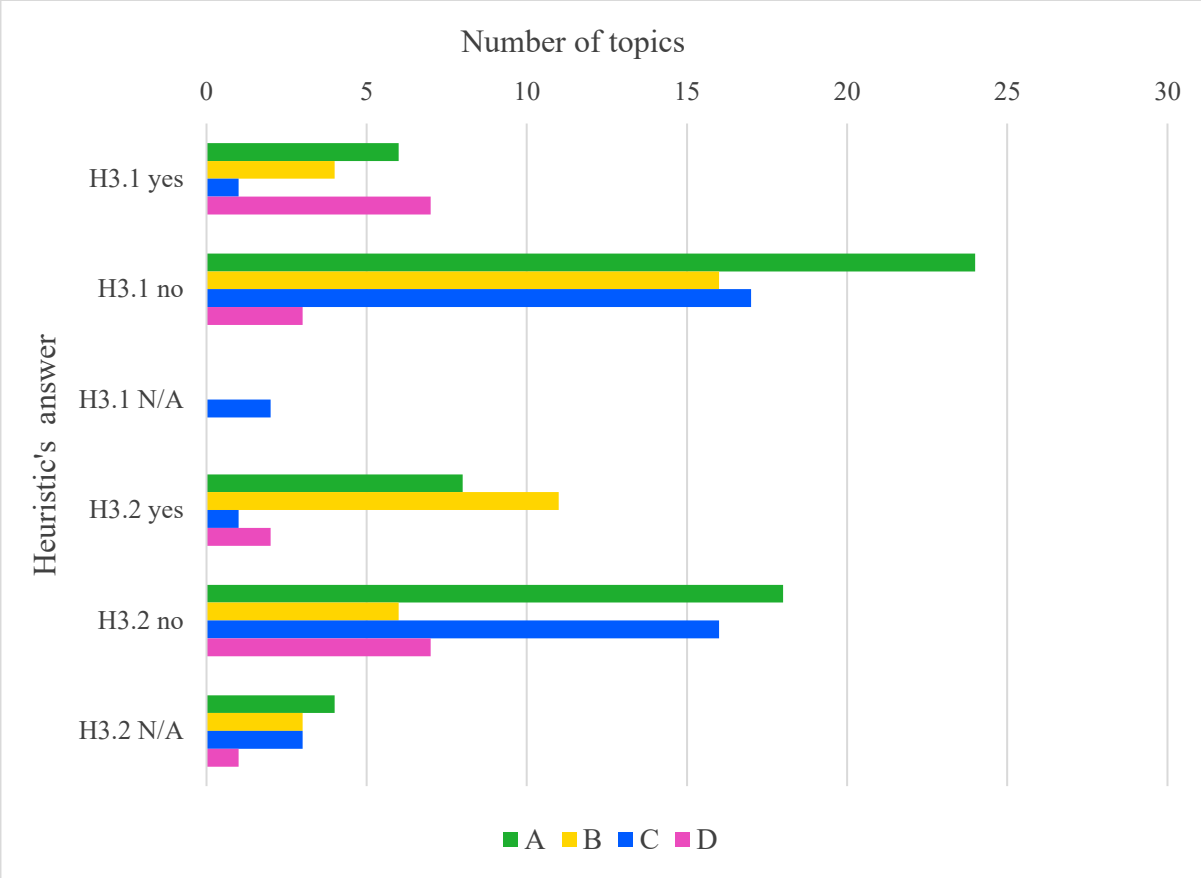
have all the available information about this subject, or is there something else that they should read, such as instructions for restoring the notification server. In addition, the above task is very brief, and it does not include any links to other topics with related information in a <prereq>, <postreq> or <related-links> element. The user could think that there is a problem with the documentation if it contradicts itself or seems defective, which affects negatively on their opinion on the documentation – or worse, if the documentation is defective, it could lead to an unwanted state in the customer system.

The unsuccessful formatting or the mismatch between the title and the content could be caused by various factors. It seems that task titles tend to follow the guideline but fail to accurately describe the real-life goal of the user. On the contrary, concept titles are more prone to be formatted as if they were tasks, which is likely because they violate heuristic 1.1 ("Does the document type definition match the information type of the topic?") as their content corresponds to the task information type, in which case they be converted into task topics. The <title> is the only compulsory element and should be crafted carefully – Self (2011, 132) recommends rewriting the title after the content of the topic is complete. Writing a draft title and revisiting it when the content is finished could be a useful way to ensure that by reading the title the user will have a clear understanding of what they can expect from the content.

5.3 Using elements

The DITA framework is simultaneously flexible and rigid – it helps reusing, reorganizing and creating content quickly, but it also calls for disciplined topic-based writing and adhering to certain principles to be able to take advantage of the benefits of DITA (Bellamy et al. 2011, ch. 1). This approach also applies to using elements appropriately and according to their design and definitions. Another reason for doing so is noted by Stevens (2018) who points out that the DITA schema enforces structural consistency – because of the rules that determine valid DITA syntax, users can anticipate where information within a topic is located, which in turn conforms to the fourth minimalist principle: support reading to do, study and locate. However, heuristics 3.1 ("Have block elements been used correctly?"), 3.2 ("Have the correct semantic inline elements been used?") and 3.3 ("Has all taggable content been tagged with semantic inline elements?") were among the most poorly performed aspects of the documentation sample – in particular heuristic 3.1, whose answer was "no" in 75% of all topics. Chart 2 shows the results for heuristics 3.1 and 3.2 and their variation in the four components.

Chart 2. Evaluation results for heuristics 3.1 and 3.2.



According to Chart 2, component A had the most issues with using both block (H3.1) and inline (H3.2) elements. The same issue is true for component C. While component B appears to fare well with using inline elements, it also had issues with block elements. Component D on the other hand had the fewest issues in both categories, but block elements were still used incorrectly in seven of its ten topics. The most common cases of incorrect use of block elements in all components were using

- instead of <steps>
- <context> or <note> instead of <prereq>
- <note> instead of <substeps> and
- <info> instead of <stepxmp> or <stepresult>.

Based on these findings, it seems that block elements are mostly used incorrectly when a more specific, appropriate element is replaced with a more general one, perhaps because the more specific element cannot be inserted due to the wrong topic type or the topic DTD syntax rules, or a lack of awareness that a more suitable element could be used. For example, several task

topics used the very general <info> element as a replacement of various more specific task elements, such as <stepxmp> for an example to illustrate a step of a task, or a <stepresult> for an explanation of the expected outcome of a step. The <info> element occurs within a <step> element, and it can nest many of the more general elements, including <note> and <codeblock>. Its purpose is to provide general additional information about the step, which is why it does not contain the same, accurate semantic meaning that the task-specific block elements do. However, <info> should only be used when the content does not fit logically into a task-specific element (Bellamy et al. 2011, ch. 2). When <info> replaces an element such as <stepxmp>, the end-result is that the incorrectly used <info> ends up containing information that does not correspond its original purpose. Another example for tasks would be a <context> element that includes prerequisite information, which should be contained in a <prereq> element.

One of the most frequent errors of using elements was using a <codeph> (an inline element) instead of <codeblock> (a block element). Both elements are used for snippets of program code, which is emphasized with a monospaced font in the output formats. The main difference in their use is that <codeph> is reserved for code that is a part of the main text flow, whereas <codeblock> is for one or more lines of code that begins in a new paragraph. In other words, their appearance has been defined to be different, which in itself is an important reason for using the elements accordingly. However, the most important differentiating factor is that unlike <codeph>, the <codeblock> element is a static block-level element that preserves the line endings, which is critical for program code when the user copies it from the documentation.

The use of semantic inline elements was problematic in slightly less than 60% of the topics, and similarly as with block elements, vague elements were used instead of the more specific ones. For example, when a procedure was written as a concept, the task-specific element <cmd> was not available in the concept DTD. In addition to the abovementioned incorrect usage, all four components had erroneously replaced other semantic inline elements with <codeph> – it was used for different types of content that describes various objects in a software, such as names of system variables (<varname>) and parameters (<parmname>), the names of files or their paths in directories (<filepath>), and input provided by the users (<userinput>). Writers undoubtedly know that these pieces of information are not program code, but in many cases the content is still tagged as <codeph>.

Based on the analysis, sometimes writers encounter a situation in which no pre-existing semantic inline element seems to reflect the content. To give an example, DITA does not have

a predefined element for different user roles, such as admin user. However, the information about which user role is in question seems to be relevant for the writers. As examples 4 and 5 show, the absence of a predefined element for the user role has resulted in varying practices in different documentation teams, as component D uses while component C prefers :

(4) (Component D, file 3)

By default, the admin and <ph keyref="[*admin_role2*]"/> users have the permission to add and remove users.

(5) (Component C, file 82)

Log in to the node as an admin user and use the [*product name*] interface to execute the following commands:

User roles are a relevant theme in all software documentation, and since some topics discuss several user roles (how they differ or what restrictions they may have), the writers might want to help the users distinguish content based on the user role it necessitates. Since the writers are already implementing this type of a strategy, it could be interesting to investigate why that is. Defining and using a unified formatting, whether that means using a highlighting element or plain text, for this purpose would also make the content more consistent as a single practice would be used instead of multiple ones (, or plain text).

According to Stevens (2018), the content may not meet the minimalist principles if it does not match the semantics of the element tag itself or if the writer is being "creative", in other word finding workarounds and using tags where they were not intended to be used. These signs signal that the content should be rewritten, restructured or omitted (ibid.). Fixing the wrongly tagged content is also important because incorrect semantic affects the output appearance and the consistency of the content. Additionally, not using the task-specific semantic markup creates inaccurate content that does not have semantic element information that could be leveraged for reuse or for defining the appearance of certain types of content. Moreover, Fraley (2003, 55) points out that if something is not marked, it cannot be searched either. If the business unit wanted to further develop its user help portals' searchability so that task-specific elements such as <stepxmp> could be searched, for instance, it would require the content to be marked up on a satisfying level beforehand. If that is not done, the documentation teams would need to add the missing markup, which in turn would mean shifting more resources to refactoring existing documentation instead of creating new content, which is not desirable.

5.4 Separating content from form

DITA-authored content does not have a native presentation format, which makes DITA the so-called archive format of the content – DITA uses semantic markup, and how the marked-up content is eventually displayed is determined outside the authoring process (Self 2011, 225). All taggable DITA content can be configured to be displayed in any specific manner in any of the possible output formats. For example, one of the most common elements in the business unit's documentation, <uicontrol>, is currently bolded in all delivery formats. However, for the content to be bolded presumes that the correct semantic tag has been used. The answers for heuristic 4.1 ("Is the topic free of highlighting elements for editorial purposes?") show that oftentimes non-semantic tags are used instead of the appropriate semantic tags: this heuristic was violated in nearly 45% of the analyzed topics. The <b> element was used incorrectly in 35% of the topics, while <i> was misused in 10% of the topics. In seven topics, <b> has replaced <uicontrol> elements. However, the use of <b> instead of <uicontrol> was sometimes arbitrary, as is the case with the fake task of component B where UI object strings were tagged with <b> and <uicontrol> almost interchangeably, with one fake step (<ol> with nested <li> elements) using <b> while the next one uses <uicontrol>. Example 6 illustrates this discrepancy:

(6) (Component B, file 32)

```
<p>To search for an object, or an [object 2], or to search for a combination of them:
   <ol>
     <li>Click the <b>Search...</b> field.<p>The <b>SCOPE</b> list opens.</p></li>
     <li>Select <uicontrol>Object selection</uicontrol>, <uicontrol>Available objects</uicontrol> or
     <uicontrol>Sort by</uicontrol> from the <uicontrol>SCOPE</uicontrol> drop-down list.</li>
   </ol>
</p>
```

Interestingly, any of the other highlighting tags (<i>, <u>, <tt>) besides <b> were not used for UI controls – therefore, it seems that if UI controls are tagged with something else than <uicontrol>, <b> is used consistently. This may be because the two main output formats of the customer documentation do bold UI controls at the moment. However, herein lies a huge risk – what would happen if it were decided to change the UI controls' appearance in the output? For example, if <uicontrol> were opted to be underlined instead of bolded, the output would be inconsistent because all UI controls are not tagged with the <uicontrol> tag set – some would be left bolded while the rest would be underlined because the publishing engine processes <b> elements differently than the <uicontrol> ones. In addition, all untagged UI controls would still be displayed without any formatting, so in the end, the documentation would include three

possible ways of formatting similar content. The inconsistent appearance of the documentation does not serve the users who may struggle with the disharmony when trying to distinguish UI controls from the body text. Furthermore, consistent formatting of content would help the users learn and recall information and therefore enhance the user experience.

A possible explanation for using instead of the appropriate semantic inline element, such as <uicontrol>, is that the writer knows by heart that the output appears as bolded, and they might think that using the bold highlighting element instead is not strictly incorrect since the output remains identical. However, this strategy will fail immediately if the formatting of <uicontrol> changes to anything other than bold. In addition, the semantic element information of <uicontrol> is lost as highlighting elements do not contain any information about the element besides how it looks. Based on the analyzed topics, semantic elements can also be overused as universal elements for different types of content with an identical appearance in the output: the monospaced <codeph> inline element was used for various software- and programming-related content that could have been tagged with other semantic inline elements such as <varname>, <parmname> or <filepath>. All of these elements are monospaced in the output formats so that they can be differentiated from the main text flow, but for some reason <codeph> is used as an "umbrella element" for other types of content than programming code.

Another way of wrongly using highlighting elements is creating fake titles by bolding plain text within a block element such as , <context> or <li> as in a concept of component C in example 7:

(7) (Component C, file 76)

There are different groups in [*Specification*] as listed below:

[*API name 1*] [*API name 1*]s are generated by [*product name*] code generator based on the [*name*] models used in the technology packs.

[*API name 2*] [*API name 2*]s are to access the actions developed in technology packs.

[*API name 3*] [*API name 3*]s are the APIs provided by [*component C*] out of the box.

The element has been used to emphasize the names of the three different application programming interfaces (APIs), and because the <li> elements contain elements that begin on a new line after the element, the output displays the text as if the bolded words were titles for the following text. However, there are three major reasons as to why this is not

advisable. First, the pieces of information in the <li> elements do not require a title – instead of being a list, the content could have been formatted as a three-column table with appropriate headers (e.g. API name, Description and Details) that are automatically formatted in a distinguishable way, which makes the content more scannable and enables quick consumption. Second, the listed API names should not be put into order as they are not sorted by their sequence or order of importance, which is the definition for using the <ol> element – if the API names need to be listed, the correct element would be <ul> ("unordered list"), which is a bulleted list for items that are not in any specific order. Third, now the API names are not tagged with the semantic inline element <apiname> that exists for the very purpose of marking up APIs. Based on the analysis, it seems that overusing highlighting elements effectively obstructs proper semantic tagging because once content is tagged – regardless of the tag –, the writer may feel that there is no need to use another tag, even though the <b> and <apiname> inline elements could also be nested. Moreover, the decision to not include an <emphasis> element in DITA was a deliberate one, and it was made with the intention to encourage writers to concentrate on what they write instead of what the writing looks like (Stevens 2018). If highlighting elements are used instead of the correct semantic elements, many benefits of DITA and semantic markup (including rapid and consistent changes to the content's appearance and more accurate search engine results) are lost, and in the long run, managing content with varying markup will cost time and effort.

According to the senior documentation specialist of the business unit, this bad practice of using <b> and <i> incorrectly has been recognized in the business unit, and at some point it was even considered that <b> elements in the source files would cause the output build process to fail, but this could not be done because there likely is an extensive amount of still used legacy content that is tagged with <b>. Avoiding this practice of replacing semantic inline elements with highlighting ones is essential for ensuring the consistent look and feel of the output formats as well as proper semantic tagging for information mining purposes.

5.5 Reusing content

Heuristic 5.3 ("Has reusable content been reused?") was for discovering whether available reusable content for the respective component has been reused. Whether the content is reusable or not depends greatly on the component in question, and as I am not an expert in any of the components, evaluating this particular guideline was challenging because I did not have enough

information to be able to tell whether certain pieces of information could or should have been reused. Altogether, this heuristic had 9 positive, 8 negative and 63 N/A answers, which meant that the vast majority topics did not have clear-cut reusable content. However, there were eight topics which should undoubtedly have reused some of their content. In six of them, the reusable content would have been the login steps for task topics. This is because in all tasks, the user should be informed of the software environment and user role which they need to use in order to accomplish a procedure because otherwise the task is not self-contained. Generally, task steps may not be easy to reuse because the steps of procedures tend to be unique in both their content and order, but there are more reuse scenarios available for product login steps and steps that describe how applications are opened, as they do not vary greatly. With the exception of one task topic, login steps were reused in component B, but the data for the other three components shows the opposite – components D, A and C did not reuse these types of steps even though some of their respective task topics within the sample included the exact same login steps, although with slightly differing wordings or DITA elements that did not change the content itself. This may be because login steps (or steps in general) have not been recognized as potential reuse cases in these components, unlike in component B, which was consistent in its reuse strategy for task topics. In addition to login steps, tasks in component B also reused other common steps, which suggests that the reuse strategy of component B is rather advanced. This finding is corroborated by the fact that eight out of the nine positive answers for heuristic 5.3 were from component B, with the ninth being from component A.

Besides reusing steps, the components differed in the extent and preciseness of how they used variables, which are referenced entities that are reused by replacing their value when the information is published (Self 2011, 178–179). The common variables mentioned in heuristic 5.1 (“Have common variables been used for respective content?”) include the proper names of the target company and the documented products, applications and portals, which are defined in a separate variables map by using keys. Being somewhat familiar with the components, I was aware of their most common proper names and could recognize them during the analysis. I also had access to the variables map and could search it to see whether a proper name has been defined as a key, which was a prerequisite for evaluating heuristic 5.2 (“Has textual content suitable to be a variable been used as one?“). For both of these heuristics, the majority of the answers was negative – 45% for heuristic 5.1 and 55% for heuristic 5.2. These numbers were mostly due to component C, which does not have a variables map and therefore does not use any variables, but also component A, whose variables strategy appears to be incoherent to some

degree. This is because certain product names that have been defined in the variables map have not been used in 12 of the component's 30 topics.

Component D was the only one that used its variables consistently, but then again a product name that occurred in three of its five tasks and that seemed to be a common one had not been defined as a variable. The documentation team may simply not have noticed that this proper name should be a variable, or perhaps it had not yet been defined in the second quarter of 2021. Nevertheless, variables should be defined as early as possible to ensure their consistent use – in addition, even though oXygen has a search-based function for tagging plain text as variables, using it could necessitate some degree of manual work, such as writing a computer script for finding and tagging only applicable content. The reason why a simple search and replace function cannot be automatically used is that proper names may also occur within program code elements (<codeph> and <codeblock>) whose content is always case-sensitive plain text in order to prevent the code from changing automatically if variables are edited, which would not be ideal in case the actual code that the users should use or see has not been changed as well.

5.6 Linking and connecting information

As Bellamy et al. (2011, ch. 1). describe, the goal of DITA is to create a web of information by linking related topics to each another by various means. Moreover, the fourth minimalism principle instructs the writers to support the user's tendency to process information in an unsystematic manner instead of the linear, page-by-page style. Based on the evaluation, it seems that instead of links to related topics, many topics exclusively rely on their order and hierarchy in the topic collection – heuristic 8.1 ("Are references within the topic links instead of plain text?") had 49 N/A answers, which means that over half of the 80 topics did not have any links to other topics. This first seems counterintuitive because DITA has a number of elements specifically for helping users navigate from one topic to another. Without clickable links, the user might need to spend time on searching for the topics in the output (HTML or PDF), which will also leave room for errors if the user cannot find the correct topic or topics. If they were given a link to the correct topic instead, they would simply need to click it to access the information immediately and continue their work. However, using links is a balancing act, as the minimalist guidelines also advise against having unnecessary links to topics that are not imperative to the topic in question. The DITA heuristics did not include heuristic for missing links, so I could not assess whether topics were missing links, per se.

Both studied topic types can have a <related-links> element at the end of the topic, which is specifically reserved for links. In this element, links to relevant topics of all types with more information on the subject are presented as a bullet list. For example, including a <related-links> section in a so-called "title topic" such as example 8 would increase the user-friendliness of the topic:

(8) (Component C, file 75)

```
<concept id="browsing_resources">
    <title>Browsing Resources</title>
    <conbody>
      <p>This section provides overview of GUI operations for [the object].</p>
    </conbody>
</concept>
```

Title topics are topics that primarily serve as an introduction to their child topics in a DITA map, and they do not contain much information besides the <title> element and possibly a brief introduction or overview in a <shortdesc> on what information will follow this topic. In the above concept's <conbody>, the phrase "overview of GUI operations" suggests that one or more tasks (GUI operations) are related to this subject – but these task topics are not elucidated anywhere in the topic, which means that the user must refer to the hierarchy of the topic collection and deduce which headings (topics) are a part of the instructions for browsing resources in example 8. However, DITA offers several more user-friendly options as it allows organizing and linking topics so that the users can effectively discover which information is related to which subject. For example, Bellamy et al. (2011, ch. 2) recommend building a set of task topics (a procedure with multiple subtasks) by creating a parent topic that describes the overall task flow, and then the child topics (the subtasks) are nested under the parent topic in a logical order. As a result, the output shows users which tasks they need to follow and in which order they must be completed, which could make the documentation of complex systems and software more transparent and easier to understand. However, it could also be argued that especially in the HTML output format, the hierarchy of the topics is visible to the users and thus there is no need to use links. Still, that necessitates an effective information model and information architecture.

The sample also included several instances of topics with plain text that should have been links created with any of the available DITA referencing mechanisms. This was the case with components A and C, as examples 9 and 10 show.

(9) (Component A, file 48)

```
<p><b>To add a [product name] plugin:</b></p>
<ol>
    <li>Download the plugin file from <xref href=[web page]</xref></li>
    <li>Create plugin directory in <codeph>[filepath]</codeph> or <codeph>[filepath]</codeph>
    where <codeph>[filepath]</codeph> is the path where user extracted the .zip file mentioned in
    <xref href="[installation topic]"/>.</li>
    <li>Extract the downloaded plugin file with <codeph>[extension]</codeph> extension to the
    directory created in step 2.</li>
```

In example 9, the third <li> element refers to the second <li> element with the phrase "in step 2" within that same topic. The phrase is plain text, and unlike properly referenced elements, it cannot be clicked to immediately access the referenced source. Using plain text also means that the text is static – it will not update without a writer manually editing it. However, numbers of steps, tables and other similar objects should not be hardcoded as plain text because in most writing environments, the writer will not know what the sequential number of the object will be until the document DITA map is processed to its delivery format (Self 2011, 161). In other words, if for any reason the procedure that is described in these steps changes so that a step is added or deleted before the third step, whoever adds or deletes that step must notice the hardcoded reference in the third <li>, and they must manually edit the step number in that phrase. As Self (2011, 161) says, there is a high risk that the hardcoded reference will not be updated. The second step (or <li> for fake tasks) could be referenced with a <xref> element so that the published output would automatically show the up-to-date number of the list item.

Similarly to example 9, a task topic of component C instructs the user to refer to a table without providing a link to it:

(10) (Component C, file 67)

```
<result>For details on the contents of the <b>[folder name]</b> folder, refer to the table
<b>[table name]</b> in <xref href="[another topic]"/>.</result>
```

In example 10, the topic in which the table resides has been linked with an <xref> element, but the name of the table is plain text tagged with <b> to highlight its name. However, the table is contained in another topic – and if the table name were changed, it surely would not be communicated to the whole documentation team because it is only a very minor detail in the whole documentation set. If the name of the table that is located in another topic changed, the person who made the change would not know that the table name has been hardcoded in the topic of example 10. Therefore, changing the table name requires a writer who edits the topic in example 10 to click the <xref> link to access that other topic with the table and see whether

the table name has been changed. In other words, no-one except the writer who edits the topic with the table would notice that the table name has been altered. Based on my experience of technical writing, writers resemble the users of documentation in the sense that they typically want to focus on a specific goal that they have, which they also want to accomplish effectively. Presumably, they would not want to waste time having to access other topics to see if a table name has been edited if that process could be automated – which is what DITA does when used correctly. From the users' point of view, it is important that the documentation is logical in its internal references, and if that is not the case, the users might begin to doubt its overall quality.

5.7 Writing minimalist and task-orientated topics

The two minimalism heuristics, 7.1 ("Is the topic free of self-evident step or task results?") and 7.2 ("Is the topic free of repetitive or unnecessary descriptive information?"), are concerned with what information is given to the user and whether that information should be given at all. As Bellamy et al. (2011, ch. 1) state, writers must understand their users' level of expertise with the product, service, or technology in terms of what the users are likely to know already and how much help they will need to resolve problems when using the product. Of course, this guideline also applies to repeating information in general because technical documentation should be succinct and quick to consume. This specific aspect was problematic in many topics, 40% of them contained repetitive or unnecessary information.

Repeating the title in one way or another was the most common instance of repetition. Even though the title would have sufficiently informed the user goal of the topic, altogether 13 topics (tasks and fake tasks) from components A, B and C essentially repeated that goal with a recommendation to follow the procedural steps via a phrase such as "To [reach the user goal of the title], follow these steps:" or "How to [reach the user goal of the title]". These prompts were located before the <steps> or <ol> element, typically either in a <context> or <shortdesc> element, neither of which is meant for this type of content. In fact, DITA does not have an element defined for this type of information, which according to Stevens (2018) suggests that the content is likely non-minimalist and should therefore be omitted or edited. There is no mentioning of including an introduction to the steps in the internal guideline either, but still it has been included in slightly over 15% of the topics. Block elements should not include these types of transition phrases because in DITA, alternative devices and a minimalist approach are used in place of transitional text, including a publishing process that can systematically apply

labels to certain parts of the output (Self 2011, 127). For example, the words "Expected outcome" are automatically rendered in the output immediately before each <stepresult> element text. Such systematic and consistent labeling allows semantic elements to be easily identified by the user (ibid.), which further increases the predictability and thus user-friendliness of the documentation, as the user can focus on their end-goal instead of understanding the documentation's structure. Bellamy et al. (2011, ch. 2) note that using or not using step introductions must be done consistently. In the business unit, the guideline is to rely on using other cues, such as using the <steps> element which is not preceded by a label in the documentation outputs, either. Instead of a label, the <step> elements within <steps> are numbered as a visual cue for signaling that the following information is step-by-step instructions that the user should follow in order. Any possible single-step instructions are bulleted and contained in a <steps-unordered> element, although the internal guideline strongly advises against using them. In addition, different topic types should have differently formatted titles (see 5.2), and therefore tasks should be recognizable by their titles with the main action verb in the *-ing* form.

One of the four principles of minimalism is task-orientation, which is related to the fact that even when users seek to understand how a product works, their intent is to exploit that information to conduct a task (Hackos 2012, 1). Minimalist instructions are also task-oriented, which means that they should support user exploration and innovation (van der Meij and Carroll 1995, 244). Therefore, writers should be extremely careful when authoring tasks – they should not include information that users do not need or care about or that they can quickly deduce. In the sample set, 10 topics (concepts and tasks in components A, B and C) that described procedures had self-evident step results that were formatted similarly to example 11:

(11) (Component C, file 79)

```
<title>Logging into the portal</title>
  <ol>
    <li>Open a web browser, type the URL of the [portal name] portal, and press
    <b>ENTER</b>.<p><b>Result</b>: The login page for the [portal name] GUI opens.</p></li>
    <li>Type in the Username and Password and then <b>SIGN IN</b>.<p><b>Result</b>:The home
    page of the portal opens.</p></li>
    <li>To log out of the portal, click the drop-down arrow next to the option <b>User</b> and select
    <b>SIGN OUT</b>.<p><b>Result</b>: Once you sign out, you are redirected to the login page of
    the portal.</p></li>
  </ol>
```

The results of steps (<li>s) 1–3 are self-evident to the expert users because as soon as they complete the command of step 1 and enter the portal's URL on a web browser, it is safe to assume that they will instantly know that the portal's login page opens as a result – this is familiar for any user who has ever logged into a portal. If there were a risk that following this command could lead to an unexpected result, perhaps then the step should include a <steptroubleshooting> element so that the user could be guided forward. However, this is not the case for any of the steps in this task, so what could explain the self-evident step results? Perhaps the intention is to help the users confirm that they are proceeding correctly. Nevertheless, ending every steps with a sentence that says that some window or page opens should be avoided, as most of the time this type of information is not needed (Bellamy et al. 2011, ch. 2), and including it makes the documentation unnecessarily lengthy. If completing a step changes the user's context and the user should be told where they are in the user interface (if the new context is unexpected or complex to understand), the new context can be embedded in the next step's <cmd> element as an introduction instead of using a <stepresult> element. Although technical writers become experts in the product, service, or technology that they document and might understand their product well, they should consider what information is truly essential for the users and omit nonessential content (ibid., ch. 1). Similarly, if the product requires explaining rather simple user operations, the product interface should be improved rather than documenting information that will probably be left unread by the users (ibid.).

Example 11 illustrates another important notion, namely that because the used topic type (concept) does not correspond to the documented content's information type (procedure), the information concerning the expected results of steps could not be contained in the correct element because <stepresult> is a task-specific element. When the incorrect topic type is used, writers may use nonconforming methods to achieve a similar result as they would have by using the correct semantic elements. Based on the sample, bolding text with <b> appears to be the most popular incorrect method. However, the achieved result is content that is not separated from form and that does not mark up information according to its meaning because highlighting elements are used instead of semantic ones. This way of working also leads to a scenario where the content does not meet the minimalism guidelines. According to Stevens (2018), certain signs can indicate if that is the case. The signs include when a writer cannot figure out which element they should use for the content or when the content does not match the semantics of the element tags – or when the writer is being creative and finding workarounds for using tags

where they were not intended (ibid.). Based on the evaluation results of the minimalism and element usage heuristics, this issue is prevalent in the analyzed sample documentation.

5.8 Summary of the evaluation

One of the major themes to be considered in this study was that DITA was designed for topic-based, task-oriented documentation, and naturally that should be reflected in the ratio of concepts and tasks where tasks should outnumber concepts. However, the analyzed documentation set included several concepts that should have been tasks, and even more topics that should have been divided into multiple smaller ones of various topic types. As the senior documentation specialist of the business unit explains, the incorrect use of the concept topic type may be explained by a haste to finish the content for a product release. In this case, the writer may not have the chance to consult the respective research and development (R&D) team to obtain more technical details or other input information for the documentation. Still, writers must bear in mind that although input from R&D will likely emphasize the product functions and features, the users have real-life goals that they want to complete, which requires task topics. Perhaps the minimalist heuristics could have been expanded with a heuristic such as "Does the topic reflect the user's core tasks instead of the system or tool?" but evaluating that aspect might have required a more profound understanding of the documentation's technical content.

In addition to preferring the task topic type in general, it should also be noted that each task should have a clear-cut, real user goal on which they focus. The documentation should provide just enough concepts to get users started, especially because very few readers have the patience to read dozens of pages of conceptual material before they start a task (Bellamy et al. 2011, ch. 1). Even though the products may be complicated, the documentation should be easy and quick to consume, and it should encourage the users to act independently. However, this applies to complex and abstract products to some extent only, as large and complex systems such as the ones that were analyzed in this study naturally require a greater amount of descriptive, conceptual information than simpler products. In this scenario, applying the minimalist principle of including the minimum amount of information that the user needs to start their task could still be followed in addition to immediately offering troubleshooting information in case of errors or providing links to related topics to guide the user to the correct path.

The usage of semantic elements was another critical part of the evaluation as along with topics, they are the foundation of DITA-authored documentation. The 611 available elements in DITA may feel overwhelming to adopt, but in practice the element pool that should be used by writers consists of 221 elements (see 3.2). The business unit has an internal DITA element guide available for all writers that includes instructions for using task-specific elements and information about which elements should not be used, but the findings of this study show that these guidelines are not followed strictly in any of the four components whose documentation was evaluated. In practice, the ways of using elements vary greatly both within and across the components' documentation sets. However, the guide is not all-encompassing, and it is possibly slightly outdated, which might partly explain the inconsistencies of element usage.

In addition to using semantic elements incorrectly, the evaluation revealed that it is rather common to replace semantic elements with non-semantic highlighting elements, especially <b> and <i>. This issue is related to one of the principles of structured writing in DITA, namely separating content from form. Highlighting elements were used in 35 topics, which is almost half of the entire sample. This problem was particularly prevalent with components A and C, with 60 and 65 per cent of their respective sample topic sets having "no" answers. The textual content that tended to be marked up with highlighting elements included various objects in the user interface, titles within the topic body, proper names and even entire steps. As the appearance of the output formats is defined in a separate process from the content creation, forcing the content to appear in a certain way by using <b> and <i> in the source format creates inconsistencies in the look and feel of the documentation. Moreover, it effectively disrupts the idea of being able to update content easily and rapidly because when the correct semantic elements are not used for the appropriate content, the content's output appearance cannot be edited with the intended DITA mechanisms. Based on the analysis, it became very clear that writers who author in DITA do affect the structure and order of the document even though DITA-authored content has been separated from form due to the automatic publishing process and other DITA mechanisms. In Fraley's real-life case study example of implementing single-sourcing at an organization, one of the established requirements for their documentation strategy was to minimize the number of ways how writers could alter style to let tagging and context determine the style instead of individual writers' choices (2003, 55), which means that the correct tags should be used instead of highlighting elements such as bold or italic. These types of established requirements are essential because they govern the documentation strategy that is designed to serve the end users of the documentation.

The evaluation results also suggest that the documentation partly conforms to the linear way of writing. Some topics were not self-contained or context-free because they did not include all necessary information (such as context about the subject of a concept or the login steps for a task), and as a consequence, they might be dependent on their location or order in the DITA map. This also reduces the reuse potential of these types of topics. Furthermore, the evaluation shows that content reuse has not been put into practice very efficiently in most of the components, since only component B reused task steps in an organized manner, while component C did not use any variables for repeated words or phrases, including the company name. In a single-sourcing environment such as DITA, using variables would allow making numerous text changes quickly and with very little effort (Self 2011, 179). Luckily, implementing a variables strategy is a relatively easy task in a single-sourcing environment.

Before starting the evaluation, I anticipated that some issues in the documentation would not be covered by any of the heuristics – and since I could not utilize a pre-existing heuristics list, I decided that I could amend the list at the beginning of the analysis. After analyzing the topics of component D, I noticed that some topic titles did not correspond to the actual content of the topic, which lead to heuristic 2.3 ("Does the title match the content of the topic body"?). Later in the analysis, I discovered that I had difficulties distinguishing between heuristic 7.2 ("Is the topic free of repetitive or unnecessary descriptive information?") and 7.3 ("Does information appear in the topic once only unless it is critical and must be repeated?"). The latter heuristic seemed to repeat the former one, which is why heuristic 7.3 was omitted from the list.

The DITA heuristics crafted for this thesis provided relevant results of each analyzed component and their topics. This was to be expected because the heuristics were created based on practical experience on the business unit's customer documentation, so some of the characteristics of the analyzed material were already recognized before the analysis. As Chart 1 (ch. 5) shows, each heuristic produced both positive and negative answers, which implies that all of the heuristics were applicable for the research data. Notably only one topic, a concept of component D, did not contradict any heuristic. Overall, the heuristics worked well for both topic types, and most of the time it was easy to determine whether the answer to the question was yes, no or non-applicable. Furthermore, dividing the heuristics by category helped to focus on the different characteristics of DITA. Sorting the answers according to their values (yes, no, N/A) proved to be a useful way for comparing both the components and the different heuristic categories in terms of which issue was the most prominent in which component, and the "open

field" notes for each negative answer provide a report that could be used for addressing the found issues.

Since the DITA heuristics were formatted on a very detailed level, it was interesting to see that even the most specific heuristics such as heuristic 8.1 ("Are references within the topic links instead of plain text?") also provided broader findings – the answers for heuristic 8.1 lead to the discovery that 60% of all topics did not have any links (either plain text or as a reference) to other topics or elements, even though linking information is one of the cornerstones of DITA. The original purpose of the heuristic was to see if links have been formatted correctly, not to map the extent to which the topics use links in general. In fact, the eight heuristic category of links and references could have included another heuristic for mapping how many topics include embedded links within the topic content. This would have been interesting to study, because as the DITA standard (OASIS 2015, 24) warns, the topic content cannot have embedded inline links if the content should be truly separated from format, structure and context because the links themselves contain context. Embedded links create dependencies upon the linked content, and the topic with the links can only make sense if the targets of each link are delivered alongside it; otherwise, the embedded link will be broken, and the content will appear as unfinished or faulty from the user's point of view. The <related-links> element would solve these issues, but it was not used in any of the analyzed topics, as using it has not been instructed in the internal guidelines.

The evaluation produced other unexpected results as well, which could be investigated further. For example, a concept of product A (file 50) was in fact formatted as if it were a task topic with instructions for configuring a resource, but on the other hand it was not a task either because it actually described how an application completes a procedure automatically without the user. As a consequence, the topic content does not correspond any of the DITA topic types, since it does not answer the question "What is?" or "How to?" because the topic contains instructions, but the user is not an active subject in the procedure. However, its title was formatted according to the task guideline, and the <shortdesc> element contains the text "How to [perform the action described in the title]", which strongly suggests that the topic contains step-by-step instructions for the user to follow. These types of unforeseen findings could provide valuable information for the business unit about its specific DITA requirements. In another research, the DITA heuristics could be further specialized into topic type-specific lists because the different information types require different types of guidelines. Naturally,

concepts do not need to be checked for self-evident task results, except if the concept is formatted as if it were a task, in which case the <ol> element would be used instead of the task-specific <steps> element, and “task results” would probably be included in a <p> element.

6 CONCLUSIONS

In this thesis, altogether 80 DITA XML concept and task topics from four different components of one business unit of the target company were studied to assess how well the customer documentation of these four components adheres to the principles of DITA authoring, structured writing, and minimalism as well as established internal guidelines of the unit. The chosen method for researching these questions was heuristic evaluation for which the DITA heuristics were created as a part of this study. The content of the analyzed DITA topics varied significantly, but they also shared common characteristics in terms of what types of issues they face and how prevalent certain themes are in the analyzed documentation.

The analysis data proved to be sufficient in both size and quality; the 19 heuristics for 80 topics equaled altogether 1520 answers which provided me with a broad view on the documentation of the analyzed components. Of course, if I had analyzed the topics in the context of the collection of topics (DITA map) instead, the results could have been even more profound, as I could have evaluated the structure and overall design of the components' documentation. Perhaps analyzing DITA topics within the context of their DITA maps could be the subject of another research. The heuristic evaluation method proved to be even more suitable for assessing the data than anticipated, perhaps because the DITA heuristics were crafted specifically for this study in the absence of an existing set of heuristics, which allowed me to customize them according to the needs of the business unit. I was also supported by the senior documentation specialist of the business unit with the heuristics, and I am confident that thanks to their helpful comments and input the heuristics enabled me to achieve research results that truly reflect the documentation. During the analysis, the importance of Nielsen's (1994b, 155–157) recommendation to have three to five evaluators became apparent, as the evaluation would have benefitted from the possibility of having in-depth discussions amongst other evaluators to confirm whether the results are accurate and justified. However, as I was the only evaluator, I ensured to revisit the topics that I could not immediately evaluate upon the first reading with a complete assurance. Still, some issues were undoubtedly left unfound, which is why conducting a heuristic evaluation with a group of writers could be a subject for follow-up research. Overall, the evaluation yielded interesting findings that reflect the real-life questions of technical documentation in large enterprises that produce complex software.

The variation between the four components' heuristic evaluation shows that there are differences in the ways in which DITA is used in the documentation teams, but certain features of DITA were challenging to all of them. The overall conclusion was that the DITA-authored content is not immaculately aligned with the design philosophies of DITA, the principles of minimalism, or the various internal guidelines of the business unit, which was to be expected. Again, as Kimber (2012, 2) states, DITA can be used in an infinite number of productive ways, but it seems that if more than one strategies are used simultaneously, the end-result is inconsistency and challenges in achieving the benefits that DITA offers. The possible explanations for the conclusions of this study could include the varying ways of working across the documentation teams, various obstacles to following the established guidelines and other practices, or the fact that the input for technical documentation often arrives at the last minute, thus risking quality at the expense of a tight schedule. It would be interesting to conduct more research into why the documentation does not conform to its requirements.

Although some of the findings may be already outdated because the research data dates back to 2021, the majority of the analyzed files are also included in the customer documentation releases in 2022. With this in mind, it is more probable that many findings still apply today. The on-going transformation project at the business unit involves directing some of the documentation teams' resources to refactoring existing content so that they could use the results of this research for addressing documentation that is currently in use. Remedying some of the recognized issues requires a rather low level of effort, while some findings necessitate more finessing. In fact, one of the major findings of this study has already been acknowledged – the documentation team of component C is currently implementing a strategy for using variables in their documentation, which is a step in the right direction in improving their DITA usage.

To produce usable products, people from the R&D organization must also participate in the customer documentation process (Virtaluoto et al. 2021, 32). Intensive collaboration between the whole organization and documentation teams might not be realistic in large companies, but I agree that the R&D plays a major role in the customer documentation process and should thus be an active and cooperative communicator with the corresponding documentation teams and writers. A technical content designer and writer at the target company notes that for the writers, close collaboration with the software developers and other stakeholders is essential – there should also be communication at the different phases of the development process, not only when the work is finished. Similarly, they add that knowing the user and their characteristics

as well as the use cases of the product are equally important for the writers. In order for the writers to have sufficient information of the complex products that they document, they should have a realistic opportunity to gain such information. Moreover, the overall attitude towards the importance of technical documentation and the competencies it requires is of the essence. Research shows that this attitude may not always be favorable due to a lack of resources or negative perspectives on technical writers' work (see e.g. Virtaluoto 2015 for such research).

As Virtaluoto et al. conclude (2021, 32), it is the user who determines the quality of the documentation. On that note, a technical content designer and writer at the target company points out that customer documentation is often a reflection of the products themselves, which means that complex products might result in complex user documentation, whereas user-friendly products are better equipped for user-centered documentation. To illustrate, they point out that the documentation for a multi-step task can be structured and organized in a task-oriented manner, but still it cannot fully hide any shortcomings in the user experience design or code implementation. Therefore, the user and their real-life tasks should be at the center of both product design and customer documentation, and based on the findings of this study, coherently following the principles of DITA and minimalism along with the internal guidelines and practices could be a step forward on the path to achieving user-centered documentation.

This case study provided thought-provoking findings about technical writing and customer documentation, especially in terms of how writers implement structured writing and semantic markup when creating content and how it affects the documentation. The business unit could use this study as supporting material for current and future development projects regarding user-centeredness, the quality of customer documentation and the writers' professional day-to-day work. From my point of view, DITA will likely continue to be an appealing documentation strategy among practitioners. Therefore, another potential research question for the technical communication field could be writers' attitudes and motivations towards DITA, as they have not yet been researched – surveying their opinions on the practicalities, benefits and challenges of writing structured documentation with semantic markup and other types of metadata could be a lucrative starting point for another research.

WORKS CITED

Data

Component A, 2021. 15 concept and 15 task DITA XML topics.

Component B, 2021. 10 concept and 10 task DITA XML topics.

Component C, 2021. 10 concept and 10 task DITA XML topics.

Component D, 2021. 5 concept and 5 task DITA XML topics.

Internal discussions with senior documentation specialist. 18 October, 2021 to 26 April, 2022.

Internal discussion with technical content designer and writer. 14 April, 2022.

Internal guideline “DITA element guideline”. Available on the target company intranet.

Internal guideline “Procedure writing guideline”. Available on the target company intranet.

References

Ament, Kurt. 2002. Single Sourcing: Building Modular Documentation. Norwich: William Andrew.

Bellamy, Laura, Michelle Carey, and Jenifer Schlotfeldt. 2011. *DITA Best Practices: A Roadmap for Writing, Editing, and Architecting in DITA*. Upper Saddle River: IBM Press. O’Reilly.

Carroll, John M. 1990. The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill. Cambridge, Massachusetts: The MIT Press.

Day, Don, Priestley, Michael, and David Schell. 2005. Introduction to the Darwin Information Typing Architecture: Toward portable technical information. IBM Press.

Eskola, Jari, and Suoranta, Juha. 1998. *Johdatus laadulliseen tutkimukseen*. Tampere: Vastapaino.

Fraley, Liz. 2003. “Beyond Theory: Making Single-sourcing Actually Work.” *Proceedings of the 21st annual international conference on Documentation* (SIGDOC ’03): 52–59. New York: Association for Computing Machinery.

Hackos, JoAnn. 1998. “Choosing a Minimalist Approach for Expert Users.” In *Minimalism Beyond the Nurnberg Funnel*, edited by John Carroll., 149–178. The MIT Press.

Hackos, JoAnn. 2004. "The information process maturity model: A 2004 update". *Best Practices,* 6,4:1–8.

Hackos, JoAnn. 2006. What is topic-based authoring? http://dita-ot.sourceforge.net/doc/ot-userguide13/xhtml/faqs/topic_authoring.html Accessed 1.3.2022.

Hackos, JoAnn. 2012. ”Minimalism Updated 2012.” 1–3. The Center for Information-Development Management. Denver, CO.

Heikniemi, Jouni. 2001. Mikä on XML? http://www.heikniemi.fi/kirj/moxml.html Accessed 13.1.2022.

Katajisto, Laura. 2020a. “Rakenteinen dokumentaatio ja DITA.” TECHS6: Rakenteinen dokumentaatio ja DITA. [Structured documentation and DITA.] Class lecture, Tampere University, Tampere, October 20, 2020.

Katajisto, Laura. 2020b. “Modulaarinen dokumentointiprosessi ja roolit.” TECHS6: Rakenteinen dokumentaatio ja DITA. [Structured documentation and DITA.] Class lecture, Tampere University, Tampere, October 27, 2020.

Katajisto, Laura. 2020c. “Informaatiomallit ja -tyypit.” TECHS6: Rakenteinen dokumentaatio ja DITA. [Structured documentation and DITA.] Class lecture, Tampere University, Tampere, November 3, 2020.

Kimber, Eliot. 2012. DITA for Practitioners. Volume I, Architecture and Technology. 1st edition. XML press.

Korvenranta, Heta. 2005. “Asiantuntija-arvioinnit.” In *Käytettävyystutkimuksen menetelmät*, edited by Ovaska, Saila, Anne Aula, and Päivi Majaranta, 111–124. Tampere University, Department of computer sciences B-2005-1.

Lyytikäinen, Jesse. 2020. Converting Technical Documentation into DITA – An Ethnographic Study of a Conversion Process. Tampere University, master’s thesis.

Isohella, Suvi. 2011. Työelämän asettamat vaatimukset teknisen viestinnän koulutuksesta valmistuneille. University of Vaasa, Licentiate thesis.

Meij, Hans van der., and Carroll, John M. 1998. “Principles and Heuristics for Designing Minimalist Instruction.” In *Minimalism Beyond the Nurnberg Funnel*, edited by John Carroll. 19–54. The MIT Press.

Nielsen, Jakob, and Molich, Rolf. 1990. Heuristic evaluation of user interfaces. ACM.

Nielsen, Jakob. 1994a. How to Conduct a Heuristic Evaluation. https://www.nngroup.com/articles/how-to-conduct-a-heuristic-evaluation/ Accessed 6.12.2021.

Nielsen, Jakob. 1994b. *Usability Engineering.* San Francisco (CA): Academic Press.

OASIS. 2015. *Darwin Information Typing Architecture (DITA) Version 1.3 Part 1: Base Edition*. Edited by Robert D. Anderson and Kristen James Eberlein. OASIS Standard. http://docs.oasis-open.org/dita/dita/v1.3/os/part1-base/dita-v1.3-os-part1-base.pdf

OASIS. 2022. About Us. https://www.oasis-open.org/org/ Accessed 18.2.2022.

OxygenXML. 2022. All DITA elements, A to Z. https://www.oxygenxml.com/dita/1.3/specs/langRef/quick-reference/all-elements-a-to-z.html Accessed 4.3.2022.

Priestley, Michael, Gretchen Hargis, and Susan Carpenter. 2001. "DITA: An XML-Based Technical Documentation Authoring and Publishing Architecture." Edited by Deborah S. Ray and Eric J. Ray. *Technical Communication* 48, 3: 352–367.

Priestley, Michael, and Swope, Amber. 2008. *DITA Maturity Model: A JustSystems white paper.* JustSystems, Inc. and IBM Corporation.

Rautava, Laura. 2018. Mobiilisovellusten käyttäjädokumentaation ominaispiirteet ja rooli käyttäjäkokemuksen luomisessa. Tampere University, master's thesis.

Rude, Carolyn D. 2009. "Mapping the Research Questions in Technical Communication." *Journal of business and technical communication*, 23:2, 174–216.

Saunders, Graydon. 2010. DITA is Hard. https://www.single-sourcing.com/dita-is-hard/ Accessed 12.1.2022.

Schengili-Roberts, Keith. "DITA, structured writing and reuse." DITAWriter (blog). April 13, 2010. https://www.ditawriter.com/dita-structured-writing-and-reuse/ Accessed 1.3.2022.

Schengili-Roberts, Keith. 2014. *DITA Diversity in Technical Documentation*. https://www.ixiasoft.com/dita-diversity-in-technical-documentation/. Accessed 16.2.2022.

Schengili-Roberts, Keith. 2018. DITA Worst Practices. Webinar 15.3.2018. https://www.infomanagementcenter.com/product/webinar-dita-worst-practices/

Schengili-Roberts, Keith. 2017. "The development of DITA XML and the need for effective content reuse." In *Current practices and trends in technical and professional communication*, edited by Stephen Crabbe, 201–223. Northallerton: Institute of Scientific and Technical Communicators ISTC.

Schengili-Roberts, Keith. 2020. A DITA-fied version of the Pilot Training Manual for the Mitchell Bomber B-25. Public Github project. https://github.com/DITAWriter/pilot_training_mitchell_bomber Accessed 11.3.2022.

Schengili-Roberts, Keith. "Companies using DITA." DITAWriter (blog). January 5, 2021. https://www.ditawriter.com/companies-using-dita/ Accessed 14.2.2022.

Self, Tony. 2009. "DITA and the Challenges of Single-Source Article Publishing." In ANZCA09: Communication, Creativity and Global Citizenship Conference, Brisbane, 2009, 253–273. Queensland University of Technology. Creative Industries Precinct. Brisbane, Australia.

Self, Tony. 2011. The DITA Style Guide: Best Practices for Authors. Scriptorium Press.

Self, Tony. 2014. Webcast: DITA Best Practices. Webinar 28.4.2014. https://www.youtube.com/watch?v=lbzmMTFlw_E

SFS-EN IEC/IEEE 82079-1 2020. Preparation of information for use (instructions for use) of products – Part 1: Principles and general requirements. Helsinki: Finnish Standards Association SFS. Accessed 17.12.2021.

Stevens, Dawn. 2018. You got DITA in my minimalism! You got minimalism in my DITA! Webinar. Comtech Services. 26.2.2018. https://www.infomanagementcenter.com/product/dita-minimalism/

Suomen teknisen viestinnän yhdistys ry (STVY). 2022. Mitä on tekninen viestintä? https://www.stvy.fi/tekninen-viestinta/ Accessed 19.4.2022.

Tekom Europe. 2022. *Defining Technical Communication*. https://www.technical-communication.org/technical-communication/defining-technical-communication Accessed 4.1.2022.

Thomas, Bob. 2014. DITA 1.3 Feature Article: Using DITA 1.3 Troubleshooting.

Virtaluoto, Jenni, Tytti Suojanen, and Suvi Isohella. 2021. "Minimalism Heuristics Revisited: Developing a Practical Review Tool." Technical Communication (Washington), vol. 68, no. 1, Society for Technical Communication, 2021, 20–36.

Wikipedia. 2022a. Help:Transclusion. Last modified April 11, 2022. https://en.wikipedia.org/wiki/Help:Transclusion Accessed 22.4.2022.

Wikipedia. 2022b. "Web page." Last modified April 21, 2022. https://en.wikipedia.org/wiki/Web_page Accessed 11.3.2022.

SUOMENKIELINEN LYHENNELMÄ

DITA XML:n käyttö ja kehitys – tapaustutkimus asiakasdokumentaatiosta

1 Johdanto

Tuotteiden käyttöohjeiden käyttäjien tarpeet ja odotukset ovat murroksessa, ja jotta niihin voitaisiin vastata entistä tehokkaammin, itse käyttöohjeet ja tavat laatia niitä ovat muuttuneet. Tuotteiden käyttöohjeiden laatimisen standardin määritelmän mukaan käyttöohjeet ovat tuotteentoimittajan kohdeyleisölle tarjoama informaatio tuotteen turvalliseen ja tehokkaaseen käyttöön sen elinkaaren aikana. Tämä informaatio kattaa käsitteet, menettelytavat ja viitemateriaalin. (SFS-EN IEC/IEEE 82079-1 2020, 14.) Tässä tutkielmassa käyttöohjeista käytetään termiä *dokumentaatio*, kun taas sen laatimiseen viitataan *dokumentointina*. Dokumentaatiota suunnittelevat ja laativat teknisen viestinnän alan ammattilaiset eli tekniset viestijät, joilta edellytetään monenlaisia kompetensseja, kuten kykyä suodattaa asiantuntijoilta saatua tietoa kohdeyleisön tarvitsemaan muotoon (STVY 2022). Alalla on havaittu, että perinteinen lineaarinen dokumentointi työpöytäjulkaisuohjelmia kuten Microsoft Wordia käyttäen ei enää riitä käyttäjäkeskeisen dokumentaation laatimiseen – sen sijaan dokumentoinnissa käytetään siihen suunniteltuja metodeja.

Eräs tällainen suosittu metodi on Darwin Information Typing Architecture eli DITA, joka on XML-pohjainen (Extensible Markup Language) arkkitehtuuri teknisen dokumentaation tuottamiseen ja julkaisemiseen. Arkkitehtuurin nimi kuvastaa sen ydinperiaatteita; ensimmäinen kirjain *D* viittaa Charles Darwinin evoluutioteoriassa esiteltyihin erikoistumiseen, adaptaatioon ja periytymiseen, kun taas *IT* liittyy informaatiotyypittelyyn, joka tarkoittaa informaation erottelemista sen käyttötarkoituksen ja semantiikan mukaan. Viimeinen kirjain *A* ilmentää DITAn olevan arkkitehtuuri eli eräänlainen laaja kokonaisuus eri työskentelytapoja ja -malleja, joita voi laajentaa tarpeen mukaan (Self 2011, 219–220; Schengili-Roberts 2017, 202). DITAn suosiota selittävät XML:n käytön tuomien etujen lisäksi sen tärkeimmät hyödyt, joihin lukeutuvat muun muassa sisällön uudelleenkäyttö, sisällönluomisen tehokkuus ja lokalisointikustannusten aleneminen (Priestley ja Swope 2008, 3). Metodin käyttäminen ei kuitenkaan ole yksinkertaista, sillä se edellyttää monenlaisia työskentelytapoja, jotka poikkeavat perinteisestä lineaarisesta teknisestä kirjoittamisesta. Tähän mennessä DITAn tutkimus on painottunut metodin käyttöönottoon ja yleisiin hyötyihin

liittyviin kysymyksiin, kun taas DITAn käytön edistämistä ei ole juurikaan tutkittu. Käytännön ongelmat on tunnistettu relevanteiksi tutkimusaiheiksi teknisen viestinnän alalla (Rude 2009, 198–199), ja tämä tutkielma pyrkii osaltaan täydentämään tätä tutkimusaukkoa.

Tämä pro gradu -tutkielma on tehty yhteistyössä globaalin yrityksen kompleksisia tuotteita valmistavan liiketoimintayksikön kanssa, jossa myös tuotteiden dokumentointi tapahtuu. Yrityksellä on hyvin kattava kokemus DITAn käytöstä, mutta yrityksen suuri koko ja sen sisäiset muutokset ovat johtaneet siihen, että DITAa ei käytetä yhtenäisellä tavalla liiketoimintayksikössä. Siksi yksikössä on aloitettu transformaatioprojekti, jonka yhtenä tavoitteena on kehittää ja yhtenäistää sen nykyisiä DITA-käytänteitä. Tämä tutkielma on osa tätä transformaatioprojektia, johon minulle tarjoutui mahdollisuus osallistua yrityksessä toteutettavan työharjoittelun ansiosta. Tutkimustehtävänäni on kartoittaa DITAn käytön nykytilannetta tarkastelemalla neljän pienen dokumentointitiimin tuottamaa teknistä dokumentaatiota lähdemuodossaan eli DITA XML -tiedostoina, joita tutkin heuristisen asiantuntija-arvioinnin avulla. Tutkimukseen sisältyy myös DITA-heuristiikkalistan luominen yhteistyössä yksikön DITA- ja dokumentaatioasiantuntijan kanssa. Tutkimuksen teoreettisen viitekehityksen muodostavat teknisen viestinnän, merkintäkielten (etenkin DITAn) sekä informaation suunnittelun ja käytettävyyden kirjallisuus. Tutkimustuloksia tullaan hyödyntämään transformaatioprojektissa, minkä lisäksi heuristiikkalistasta kehitetään työkalu liiketoimintayksikön teknisille viestijöille heuristiikkoja laajentamalla.

2 Merkintäkielet ja DITA

XML on World Wide Web Consortiumin vuonna 1998 julkaisema SGML-standardiin perustuva metakieli eli standardi merkintäkielten, kuten DITAn luomiselle. Merkintäkielet ovat sekä ihmisen että koneen luettavissa, ja ne käyttävät metainformaatiota kuvatakseen joko tekstin rakennetta tai esitysmuotoa. XML:n avulla voidaan kirjoittaa rakenteista tekstiä, joka tehdään XML-merkintäkielillä, kuten DITAlla. Rakenteinen teksti koostuu elementeistä, joiden sisällä on joko tekstiä tai toisia elementtejä. Elementtien nimissä kaikki sanat paitsi ”xml” ovat sallittuja. (Katajisto 2020a.) Elementit muotoillaan alku- ja lopputägeistä, jotka kirjoitetaan < (pienempi kuin) ja > (suurempi kuin) -merkeillä. Lopputägi eroaa alkutägistä siten, että siinä on kauttaviiva (/). Tägit ovat myös merkkikokoriippuvaisia. Esimerkit 1 ja 2 havainnollistavat eron oikean ja väärän tägäyksen välillä.

(1) <esimerkki>Tämä elementti on muodostettu oikein.</esimerkki>
(2) <esimerkki>Tämä elementti on muodostettu väärin.</Esimerkki>

Elementeillä voi olla myös attribuutteja, kuten @importance, jotka sisältävät lisätietoa elementeistä ja joiden avulla sisältöä voidaan suodattaa konditioiden mukaan. Monien XML:ien syntaksi on semanttista, eli elementtien ja attribuuttien nimet kertovat niiden merkityksen eli mitä ne ovat. Esimerkkien 1 ja 2 tapauksessa kummankin elementin nimi on <esimerkki>, sillä niiden sisältämät tekstit ovat esimerkkejä. Rakenteisessa tekstissä informaation sisältö ja muotoilu on erotettu toisistaan – toisin kuin esimerkiksi HTML-pohjainen teksti, XML:llä kirjoitettu rakenteinen teksti ei kerro *millaista* sisältö on, vaan sillä kerrotaan *mitä* sisältö on. Sisällön ulkonäön määrittää erillinen tyylitiedosto, kuten CSS (Cascading Stylesheet) tai XSL (Extensible Style Language). Koska rakenteista tekstiä harvoin luetaan sellaisenaan, tarvitaan XSLT-tiedosto (Extensible Stylesheet Language Transformations), jotta teksti voidaan muuntaa koneellisesti johonkin julkaisumuotoon, kuten PDF:ksi tai HTML:ksi. Elementeille voidaankin määrittää sopiva muotoilu kussakin julkaisumuodossa. Jotta rakenteista tekstiä voidaan julkaista, täytyy käyttää XML:ää sekä XML-editoria, minkä lisäksi tarvitaan tapa julkaista XML:ää (Katajisto 2020a). Tässä tutkimuksessa XML-editori on dokumentointitiimien käyttämä Oxygen XML Author eli oXygen. Lisäksi kukin XML-dokumentti vaatii DTD:n (Document Type Definition), joka on rakenteisen tekstin ”kielioppi” eli syntaksi, joka sisältää XML-tiedoston säännöt, kuten mitä elementtejä siinä voi käyttää ja missä järjestyksessä sekä mitkä elementit ovat pakollisia ja mitkä valinnaisia. XML-tiedostojen tulee myös olla oikein muodostettuja (*well-formed*) ja valideja (*valid*), eli niiden täytyy olla rakennettu XML:n yleisen syntaksin ja DTD:nsä mukaisesti. (Heikniemi 2001.)

XML on suosittu tapa laatia dokumentaatiota, sillä sen avulla voidaan saavuttaa kustannussäästöjä vähentyneiden lokalisointi- ja kirjoittamistarpeiden ansiosta. Lisäksi sisällön laatu paranee, sillä XML:n jäykät säännöt pakottavat vähintään tietynasteiseen yhdenmukaisuuteen ja johdonmukaisuuteen. (Katajisto 2020a.) Teknisen viestinnän alalla yksi tämän hetken suosituimmista XML-standardeista on DITA XML, jossa yhdistyy useita teknisen viestinnän kannalta merkittäviä osa-alueita rakenteisen tekstin periaatteiden lisäksi. DITA sai alkunsa 2000-luvun alussa, jolloin IBM kehitti sen omiin dokumentointitarpeisiinsa ja myöhemmin vuonna 2004 siitä tuli OASIS-yhteenliittymän ylläpitämä avoin standardi. DITA-standardin ensimmäinen versio julkaistiin vuonna 2005, ja kymmenen vuotta myöhemmin julkaistu versio 1.3 on tällä hetkellä viimeisin versio, jota myös analysoitavat komponentit käyttävät. DITAn yhteydessä julkaistiin myös DITA Open Toolkit (DITA-OT), jonka avulla DITA XML -dokumentteja voi muuntaa eri julkaisumuotoihin, kuten HTML:ksi ja PDF:ksi. DITAn ensimmäisten käyttäjien joukossa oli eritoten suuria teknologiayrityksiä, kuten Adobe

ja Siemens Medical, joilla oli resursseja hyödyntää DITAa edistyksellisesti, ja ajan myötä näiden yritysten luomat käytänteet levisivät eri aloille (Schengili-Roberts 2014; 2017, 205). On myös arvioitu, että DITA on yksi nopeiten kasvavia rakenteellisen kirjoittamisen standardeja (Schengili-Roberts 2017, 201).

DITA on kokonaisvaltainen ratkaisu tuottaa dokumentaatiota, mutta sen lisäksi se on *semanttinen* merkintäkieli. Sisältö merkitään semanttisilla elementeillä, jotka kuvaavat mitä kyseinen informaatio on – DITA-elementin <title> nimi kertoo sen sisältävän otsikon. DITAssa informaation sisältö on erillään sen muodosta, mikä tarkoittaa, että kaikenlainen konteksti (tekstuaalinen ja sisällöllinen) tulee häivyttää, sillä DITA on niin sanottu arkistoformaatti, ja dokumentin lopullinen muoto määräytyy erillisessä prosessissa (Self 2011, 225–229). Näin ollen elementit eivät kuvaile esimerkiksi tekstin ulkonäköä, toisin kuin vaikkapa HTML:ssä.

DITAn arkkitehtuuri perustuu *topiikkeihin* (*topics*), joita kutsutaan myös moduuleiksi. Topiikki on yksittäinen idea tai aihe, joka muodostaa oman kokonaisuutensa. Kuten DITA-standardissa kuvataan, topiikki voi olla vain otsikkonsa tai yhden kappaleen mittainen, ja sen sisältö voi olla luonteeltaan yleisluontoista tai hyvinkin spesifiä. Tyypillisesti yksi topiikki kattaa tarkkaan rajatun aiheen itsenäisenä kokonaisuutena, joka on vain tarvittavan pituinen, ja yhtä topiikkia voi käyttää useissa eri konteksteissa. (OASIS 2015, 26.) DITA-dokumentit muodostuvat mistä tahansa joukosta topiikkeja, jotka on sijoitettu *DITA-mappiin*, joka määrittelee topiikkien järjestyksen ja hierarkian (Self 2011, 29). DITA sisältää kaiken tarvittavan modulaariseen sisällöntuotantoon, ja yritykset voivat ottaa sen käyttöön sellaisenaan tai tarvittaessa muokata mieleisekseen *spesialisoinnin* avulla (Katajisto 2020a). Modulaarinen kirjoittaminen sai Hackosin mukaan alkunsa 1990-luvun puolessa välissä, kun teknisen viestinnän alalla havahduttiin sähköisiin dokumentaatioportaaleihin, eikä olemassa olevaa lineaarista ja kirjamaista dokumentaatiota ollutkaan mahdollista jakaa yksittäisiksi moduuleiksi, jotka toimisivat sähköisessä muodossa. Modulaarisuus on yhä suosittu toimintatapa, koska se mahdollistaa nopeamman sisällön tuottamisen ja kääntämisen sekä säästää siten kustannuksia.

DITAssa modulaarisuuteen liittyy myös informaatiotyypittely eli sisällön jakaminen topiikkeihin sen perusteella, mihin käyttäjien kysymyksiin ja tarpeisiin informaatio tarkalleen vastaa. Informaatio jaotellaan eri topiikkityyppeihin, joita DITAssa oli alun perin kolme: konsepti (*concept*), tehtävä (*task*) ja referenssi (*reference*). (OASIS 2015, 26.) Konseptit kertovat ja kuvailevat, millaista tai mitä jokin on, jotta sen voi ymmärtää, kun taas tehtävät neuvovat tarkat, vaiheittaiset ohjeet eli proseduurin siihen, miten jokin tehdään. Referenssit

puolestaan ovat sellaista spesifiä informaatiota varten, jota käyttäjä ei yleensä tarvitse tai voi muistaa ulkoa, mutta voi käydä satunnaisesti katsomassa tarpeen mukaan. Myöhemmin DITA 1.3:een lisättiin neljäs topiikkityyppi, vianetsintä (*troubleshooting*) (Thomas 2014). Darwinin evoluutioteoriaan nojaten nämä topiikkityypit ”periytyvät” geneerisestä topiikista nimeltä ”topiikki” (*topic*). Jokaista topiikkityyppiä vastaa samanniminen DTD. Näin ollen kullakin informaation ”palalla” on tietty tarkoitus sekä omat sääntönsä ja erikoispiirteensä, jotka erottavat sen muista (Katajisto 2020c). Informaatio jaetaan moduuleiksi, jotta käyttäjä voi hakea juuri sen tiedon, jonka hän tarvitsee voidakseen suoriutua tehtävästään. Jos käyttäjä haluaisi esimerkiksi säätää moottoripyöränsä venttiilejä, hän tuskin haluaa lukea syvällistä kuvausta niistä, vaan saada tarkat ohjeet itse proseduuriin; tässä tapauksessa käyttäjä luultavasti haluaisi luettavakseen ohjeen, joka sisältää vaiheittaisen ohjeistuksen tehtävän suorittamiseksi ja mahdolliset varoitukset vaaratilanteista, sekä tiedon mahdollisesta seuraavasta tehtävästä. Näin ollen yksittäinen tehtävä tulee esittää tehtävälähtöisestä näkökulmasta, eikä tehtävän ohella tulisi antaa siihen kuulumatonta informaatiota. Tämä periaate liittyy John Carrollin 1990-luvun alkupuolella kehittämään *minimalismiin*.

Teknisessä viestinnässä tunnettu minimalismi on käyttäjä- ja toimintakeskeisyyttä painottava oppi, jonka Carroll kehitti vastalauseena niin kutsutulle systemaattiselle lähestymistavalle (*systems approach*), jossa käyttäjien tehtäviä kohdellaan systemaattisina kokonaisuuksina, jotka voi esittää pienten palasten muodostamina oppimisvaiheina käyttäjän osaamisesta ja lähtötasosta riippumatta. Carroll sen sijaan esitti, että dokumentaation tulisi tukea käyttäjän luontaista oppimiskykyä ja -halua sekä kunnioittaa käyttäjien loogista päättelykykyä, sillä se paitsi tukee käyttäjää myös tarjoaa tälle mahdollisuuden jäsennellä ja soveltaa tietotaitoaan. (Carroll 1990, 73–75.) DITA mahdollistaa minimalistisen dokumentaation tarjoamalla siihen sopivan viitekehyksen, mutta ei pakota siihen (Stevens 2018).

Ehkäpä DITAn merkittävin etu on sen sisäänrakennetut uudelleenkäyttömekanismit conref, keyref ja conkeyref. DITA mahdollistaa kaikenkokoisten tiedonpalasten uudelleenkäytön aina koko topiikista yksittäiseen elementtiin. Sisällön uudelleenkäyttö tarkoittaa jo kirjoitetun tekstin käyttämistä uudelleen sellaisena kuin se on ilman muokkaamista. Uudelleenkäyttö edellyttää systemaattista strategiaa, ja oikein käytettynä se tehostaa dokumentointia tehden siitä myös yhtenäisempää, minkä lisäksi dokumentoinnin kustannukset laskevat (Bellamy et al. 2011, luku 1). DITAssa uudelleenkäyttö kytkeytyy yksilähteistämiseen (*single-sourcing*), joka tarkoittaa modulaarisen sisällön tuottamista yhdessä tietokannassa eli lähteessä, josta

dokumentit kootaan. Kun lähteen päivittää, päivittyvät kaikki sen esiintymät riippumatta siitä, missä mapeissa tai topiikeissa ne sijaitsevat. Jotta topiikit olisivat vielä uudelleenkäytettävämpiä, niissä voi käyttää tekstin julkaisuhetkellä prosessoitavia variaabeleita ilmaisemaan tuotteittain vaihtelevaa informaatiota, kuten vaikkapa tuotenimiä.

Vaikka DITAssa onkin monia etuja, sen oikeaoppinen käyttäminen ja etenkin käyttöönotto voi olla haasteellista, sillä DITAn käyttö eroaa monella tapaa lineaarisesta kirjoittamisesta ja monista muista tutuista työtavoista. Sisällön luominen ja sen ulkonäön määrittäminen ovat ensinnäkin täysin erillisiä prosesseja, mikä voi tuntua informaation visualisointiin tottuneesta teknisestä kirjoittajasta normaalista poikkeavalta. Toisaalta DITAa käytetään tyypillisesti ympäristöissä, joissa informaation tehokkuus, oikeellisuus ja yhdenmukaisuus ovat ulkonäköön vaikuttamista tärkeämpiä arvoja. (Saunders 2010.) DITA ei välttämättä sovi kaikille, kuten pienille toimijoille tai yrityksille, jotka eivät lokalisoi sisältöään tai eivät hyötyisi merkittävästi sisällön uudelleenkäytöstä. Toisaalta DITAn käytön voi aloittaa kevennetysti valikoimalla tietyt ominaisuudet, ja arkkitehtuuri onkin suunniteltu siten, että DITA-strategiaa voi kasvattaa vähitellen sitä mukaa kun sen käyttötarkoitukset kehittyvät (Priestley ja Swope 2008, 3).

3 Heuristinen asiantuntija-arviointi ja DITA-heuristiikat

Tässä tutkimuksessa käytän arviointimenetelmänä heuristista asiantuntija-arviointia. Heuristinen arviointi on Jakob Nielsenin kehittämä käytettävyystutkimuksen menetelmä, joka perustuu heuristiikkojen eli erilaisten käytettävyysperiaatteiden, sääntöjen tai ohjeistuslistojen käyttöön. Arviointi voidaan tehdä missä tahansa tuotekehityksen vaiheessa ilman testikäyttäjiä, ja arvioinnin tekee asiantuntija. (Korvenranta 2005, 111–113.) Nielsen itse kuvaa heuristista arviointia ”halpakäytettävyysmenetelmäksi”, sillä se on joustava eikä siihen liity tiukkoja raameja, hintavia työkaluja tai testikäyttäjien tuomia kustannuksia. Hänen mukaansa tutkimusmenetelmän ollessa kevyt on jopa todennäköisempää, että käytettävyyttä ylipäätään tutkitaan, kun siihen ei tarvitse käyttää huomattavia resursseja. (Nielsen 1994b, 17.)

Heuristisen arvioinnin tekee yksi tai useampi arvioija, joka tarkastelee tutkimuskohdetta itsenäisesti ja systemaattisesti merkiten muistiin sen onnistumiset ja puutteet heuristiikkoja hyödyntäen. Merkinnöistä tulisi käydä ilmi havainnon lisäksi se, mitä heuristiikkaa rikotaan ja miten se ilmenee. Jossain arvioinneissa saattaa lisäksi olla aiheellista merkitä rikkomuksen vakavuus käyttäen siihen tarkoitettua asteikkoa. (Nielsen ja Molich 1990, 249.) Tutkimusten perusteella kolmesta viiteen arvioijaa löytää suurimman osan puutteista, kun taas lisäarvioijista

on verrattain vähän lisähyötyä (Nielsen 1994b, 155–157). Tässä tutkimuksessa olen ainoa arvioija, mikä tulee huomioida tutkimustuloksissa. En myöskään voi toteuttaa Nielsenin (mt.) suosittelemaa prosessia, jossa arvioijat kokoontuvat lopuksi keskustelemaan löydöksistään ja vertailemaan niitä keskenään. Nielsenin mukaan (1994a) yksittäinen arvioija löytää arviolta 35 % kaikista ongelmista, eli tilastollisesti en voinut havaita kaikkia seikkoja.

Valmiita heuristiikkalistoja onkin olemassa useita. Yksi teknisen viestinnän alan tuoreimmista esimerkeistä lienee Virtaluodon, Suojasen ja Isohellan uusittujen minimalismiheuristiikkojen lista (2021), joka pohjautuu Carrollin ja van der Meij'n alkuperäisiin minimalismiheuristiikkoihin vuodelta 1995. Tutkimusmenetelmän joustavuuden ansiosta heuristiikkoja on laadittu hyvin erilaisiin tarkoituksiin, kuten videopelien (ks. Pinelle, Wong ja Stach 2008) ja käännösten (ks. Suojanen ja Tuominen 2015) arvioimiseen. Olemassa olevien heuristiikkojen käytön lisäksi on myös mahdollista räätälöidä oma lista, kuten Laura Rautava teki pro gradu -tutkielmassaan (2018). Korvenrannan mukaan arvioinnista saadaan paras hyöty, kun heuristiikat laaditaan tuotekohtaisesti, vaikka prosessi onkin työläs. Toisaalta kerran suunniteltua listaa voidaan hyödyntää iteratiivisesti tuotteen sisällä uusissa arvioinneissa. (Korvenranta 2005, 122–123.)

Heuristista arviointia on kritisoitu samasta syystä, jonka takia menetelmää toisaalta pidetään erityisen hyödyllisenä – tuotteen loppukäyttäjät eivät ole mukana prosessissa. Toisaalta käyttäjiä ei välttämättä haluta ottaa mukaan kaikkiin tutkimuksiin. Esimerkiksi tämän tutkimuksen puitteissa loppukäyttäjien hyödyntäminen ei olisi tarkoituksenmukaista, sillä arvioinnin kohteena ei ole dokumentaation lopullinen ilmenemismuoto, vaan sen lähdeformaatti DITA XML, jota käyttäjät eivät näe lukiessaan dokumentaatiota PDF- ja HTML-julkaisumuodoissa. Loppukäyttäjät eivät myöskään ole teknisen viestinnän alan asiantuntijoita, jolloin heidän erityisosaamisensa ei soveltuisi tutkimustehtävään. Tutkimuksen tavoitteena on kuitenkin tunnistaa dokumentaatiosta piirteitä, jotka aiheuttavat loppukäyttäjille käytettävyysongelmia, joten loppukäyttäjät ovat tällä tapaa olennainen osa tutkimusta.

Tässä tutkimuksessa tarkastelen 80 DITA-topiikkia yksitellen ja ilman niitä ympäröivää kontekstia eli niiden DITA-mappeja. Esitän jokaiselle topiikille samat 19 kysymysmuodossa olevaa heuristiikkaa ja kirjaan havaintoni ylös. Kuhunkin heuristiikkaan on mahdollista vastata kolmella eri tavalla: jos heuristiikassa esitelty ohjeistus toteutuu topiikissa, on vastaus ”kyllä”. Jollei ohjeistus toteudu, vastaus on ”ei”. Mikäli heuristiikan kysymystä ei voi esittää topiikille (jos topiikista puuttuu kysymyksen edellyttämä sisältö), vastaus on ”N/A” (*non-applicable*).

Vastausten lisäksi raportoin mahdollisuuksien mukaan miksi heuristiikkaa on rikottu, mikä voisi olla syy rikkomukseen ja miten ongelman voisi ratkaista.

Seuraavaksi esittelen tätä tutkimusta varten luodut DITA-heuristiikat, joiden suunnittelu oli osa tutkimusta. Laadin heuristiikat yhteistyössä yksikön dokumentaatioasiantuntijan kanssa. Niiden taustalla vaikuttavat teknisen viestinnän alan kirjallisuuden ohella liikeyksikön sisäiset dokumentointi- ja DITA-ohjeistukset. Lisäksi hyödynsin niin teoreettista kuin käytännönkin tietämystäni teknisestä viestinnästä ja rakenteisesta kirjoittamisesta. Olin myös saanut yleiskuvan analysoitavien komponenttien dokumentaatiosta oltuani teknisen kirjoittamisen harjoittelijana yrityksessä, mikä auttoi heuristiikkojen valitsemisessa ja muotoilemisessa.

Heuristiikoissa keskitytään konsepteihin ja tehtäviin liittyviin kysymyksiin, sillä tutkimus rajautuu näihin kahteen topiikkityyppiin. Heuristiikka 7.1 pätee ainoastaan tehtäviin. Lista etenee siinä järjestyksessä, missä topiikitkin tyypillisesti etenevät. Heuristiikat on jaettu kahdeksaan kategoriaan, jotka edustavat analysoitavalle dokumentaatiolle olennaisia DITAn pääpiirteitä. Ensimmäinen kategoria viittaa siihen, miten topiikkityyppi tulisi valita informaatiotyypin mukaan, sillä tämä käytäntö yhtenäistää dokumentaatiota ja tekee siitä ennustettavampaa, kun käyttäjät tottuvat eri topiikkityyppien rakenteisiin. Toinen kategoria käsittelee otsikoiden muotoilua, ne esittelevät topiikin ja toimivat navigaation apuvälineinä dokumenteissa. Lisäksi eri topiikkityyppien otsikoinnille on laadittu omat ohjeensa. Kolmas kategoria tarkastelee, onko elementtejä käytetty oikein ja niiden semanttisen merkityksen mukaisesti – DITAssa on 611 elementtiä (OxygenXML 2022), joista komponenttien dokumentointitiimit käyttävät 221 yksikön liiketoiminta-alueeseen kuuluvaa elementtiä. Neljäs kategoria keskittyy sisällön erottamiseen muodosta, mikä on edellytys sisällön tehokkaalle yksilähteistämiselle ja uudelleenkäytölle. Vaikka DITAssa onkin määritelty tyylielementit <b> (**lihavointi**), <i> (*kursiivi*), <u> (<u>alleviivaus</u>) ja <tt> (`tasalevyinen`), niitä ei tule käyttää, ellei soveltuvaa semanttista elementtiä ole. Viides kategoria liittyy sisällön uudelleenkäyttöön niin uudelleenkäytettävyyden kuin variaabeleiden käytön osalta. Kuudes kategoria nivoutuu modulaarisuuden kysymyksiin, kuten topiikkien itsenäisyyteen, tarkoituksenmukaisuuteen ja laajuuteen. Seitsemäs kategoria avaa minimalismin periaatteita, jotka ohjeistavat keskittymään käyttäjän ydintehtäviin tuotteen ominaisuuksien selostamisen sijaan. Samaten minimalismi liittyy turhan tekstin karsimiseen – ennalta-arvattavien tai käyttäjälle välittömästi selviävien lopputulosten sijaan olisi tärkeää dokumentoida odottamattomat ja käyttäjän kannalta kriittiset sisällöt. Kahdeksas kategoria kattaa DITAn sisäiset linkittämis- ja viittaustekniikat.

Taulukko 1. DITA-heuristiikat.

NUMERO	HEURISTIIKKA
1	**INFORMAATIOTYYPITTELY**
1.1	Vastaako käytetty DTD topiikin informaatiotyyppiä?
2	**OTSIKOINTI**
2.1	Onko otsikossa enemmän kuin yksi sana?
2.2	Ilmeneekö topiikin otsikosta topiikin sisältö riittävällä tarkkuudella (konseptit), tai kerrotaanko siinä käyttäjän varsinainen tavoite (tehtävät)?
2.3	Vastaako topiikin otsikko topiikin sisältöä?
2.4	Vastaako topiikin otsikko käytetyn topiikkityypin ohjeistusta?
3	**ELEMENTTIEN KÄYTTÖ**
3.1	Onko lohkoelementtejä käytetty oikein?
3.2	Onko käytetty oikeita semanttisia sisäelementtejä?
3.3	Onko tägättäväksi soveltuva sisältö tägätty semanttisilla sisäelementeillä?
4	**SISÄLLÖN JA MUODON EROTTELU**
4.1	Onko topiikki välttynyt sen ulkoasuun vaikuttavilta korostuselementeiltä?
4.2	Onko topiikki välttynyt tyhjiltä elementeiltä?
5	**UUDELLEENKÄYTTÖ**
5.1	Onko yleisiä variaabeleita käytetty tarkoituksenmukaisesti?
5.2	Onko variaabeliksi soveltuvia tekstisisältöjä käytetty variaabeleina?
5.3	Onko uudelleenkäyttöön soveltuvaa sisältöä käytetty uudelleen?
5.4	Onko topiikki välttynyt siirtymäsanoilta informaation välillä?
6	**MODULAARISUUS**
6.1	Onko topiikilla yksi selvästi tunnistettavissa oleva tavoite?
6.2	Muodostaako topiikki itsenäisen kokonaisuuden?
7	**MINIMALISMI**
7.1	Onko topiikki välttynyt itsestäänselviltä lopputuloksilta (tehtävät ja vaiheet)?
7.2	Onko topiikki välttynyt toistuvalta tai tarpeettomalta kuvailulta?
8	**LINKIT JA VIITTAUKSET**
8.1	Ovatko topiikissa esiintyvät viittaukset muihin sisältöihin linkkejä muotoilemattoman tekstin sijaan?

4 Tutkimusaineisto

Tässä tutkimuksessa teen heuristisen arvioinnin liiketoimintayksikön neljän tuotekomponentin (A, B, C ja D) dokumentaatiolle, jonka laatii komponentin oma dokumentointitiimi. Dokumentaatio toimitetaan HTML- ja PDF-muodoissa asiakasportaalissa sekä ohjelmistoon integroituina ohjeina. Nämä komponentit valikoituivat tutkimukseen, sillä ne ovat mukana transformaatioprojektissa. Tutkimusaineisto koostuu 80 DITA-topiikista, joista puolet on concept-topiikkeja ja puolet on task-topiikkeja. Tutkimusaineistolle oli kaksi kriteeriä: sen pitää olla julkaistu osana tuotejulkaisua (jolloin topiikit on toimitettu loppukäyttäjille validina dokumentaationa), ja julkaisun tulee ajoittua ajalle ennen transformaatioprojektia. Näin ollen ajankohdaksi valikoitui vuoden 2021 toinen neljännes. Aineisto on kerätty teknisten kirjoittajien käyttämän Git-työkalun avulla. Git on hajautettu versionhallintajärjestelmä, jonka avulla usean henkilön on mahdollista työstää projektia lokaalisti ja samanaikaisesti tietokannassa eli *repositoriossa*. Git tallentaa kopioita projektin versioista, mikä mahdollistaa aikaisempiin versioihin palaamisen ja niiden tarkastelun. Tämän ansiosta pystyin lataamaan paikalliset kopiot vuoden 2021 toisen kvartaalin aikaisista repositorioista.

Koska tämä tutkielma edustaa laadullista tutkimusta, minun tuli pohtia miten määrittäisin tutkimusaineiston koon. Eskolan ja Suorannan (1998, 61–62) mukaan laadullisessa tutkimuksessa ei ole ennalta määrättyä aineistokokoa, eikä aineiston koko myöskään ratkaise tutkimuksen onnistuneisuutta. Aineistoa on oltava riittävä määrä, mutta toisaalta liian suurta aineistoa ei ole mahdollista tutkia riittävän kattavasti. Tietääkseni DITA XML -lähdeformaatin dokumentaatiota ei ole aiemmin tutkittu heuristista arviointia käyttämällä, joten en voinut perustaa arviotani aineiston koosta aiempaan tutkimukseen. Niinpä laskin neljän analysoitavan komponentin topiikkien kokonaismäärän ja niiden jakauman topiikkityypeittäin arvioidakseni, kuinka monta topiikkia minun tulisi analysoida. Taulukko 2 näyttää laskelman lopputuloksen.

Taulukko 2. Komponenttien topiikit ja niiden jakauma.

Tuotejulkaisu vuoden 2021 toisella kvartaalilla					
Komponentti	**Topiikkien määrä**		**Topiikkien määrä komponenteittain**	**Osuus (%) kaikista**	**Analysoitavien topiikkien määrä (50% konsepteja, 50% tehtäviä)**
	konsepti	**tehtävä**			
A	1226	161	1387	41	30
B	539	303	842	25	20
C	773	124	897	26	20
D	101	169	270	8	10
yhteensä	2639	757	3396	100	80

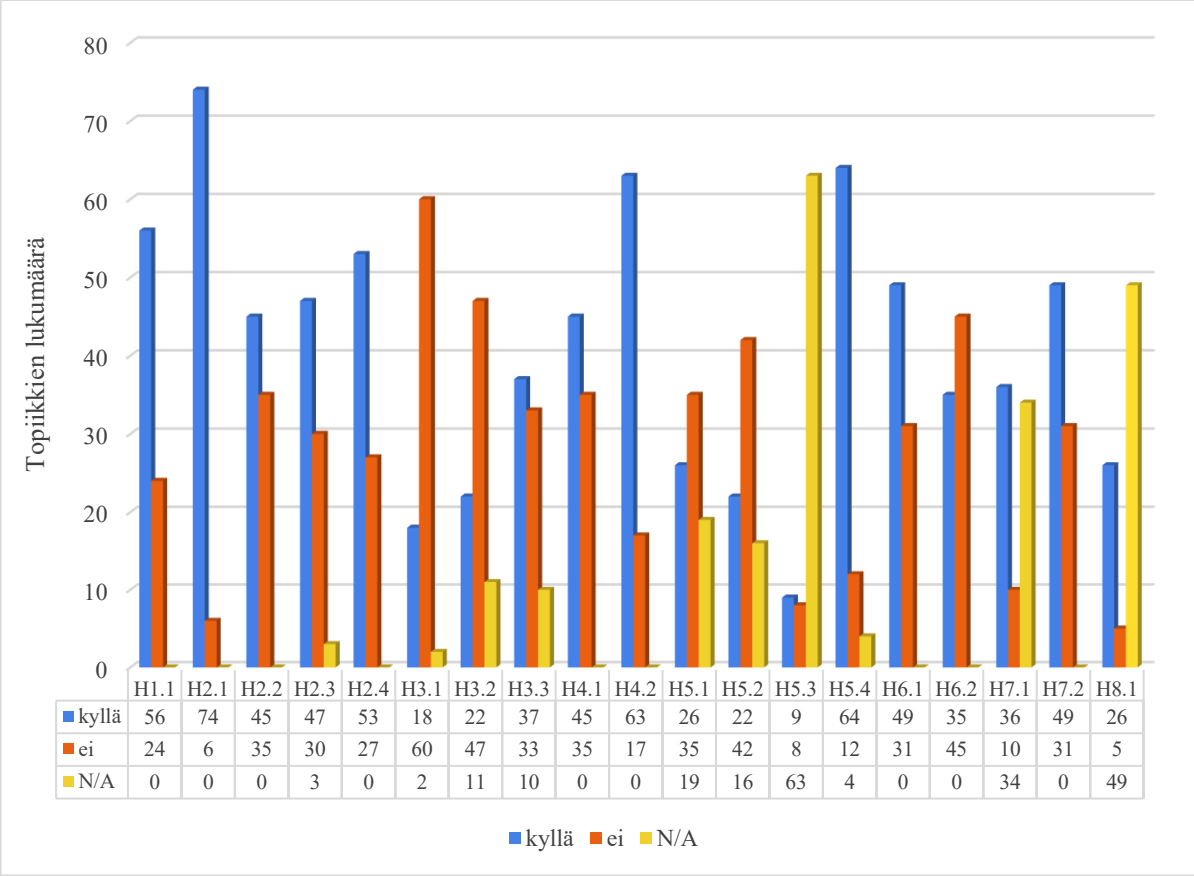
Topiikkeja on siis yhteensä noin 3400, tosin osa niistä saattaa olla tarpeettomia tai sisällöltään kaksoiskappaleita. Vaikka yksittäinen topiikki saattaakin olla vain otsikkonsa mittainen, en voisi pro gradu -tutkielman puitteissa analysoida tuhansia topiikkeja, enkä välttämättä edes satoja. Toisaalta laadullinen aineisto voi saturoitua eli kylläänty, kun aineisto lakkaa tuottamasta uusia tutkimustuloksia (Eskola ja Suoranta 1998, 62–63). Kaikkien DITA-topiikkien on noudatettava niiden DTD:iden kielioppia ollakseen valideja, jolloin tutkittavien topiikkien voi olettaa kylläänty vän tietyn pisteen jälkeen. Mutta miten kylläänty mispisteen voi määrittää? Eskolan ja Suorannan mukaan (1998, 62–63) sen voi määrittää myös tutkimuksen edetessä, jolloin aineiston kokoa voisi kasvattaa, kunnes koetaan, että kylläänty mispiste on saavutettu. Laadullinen tutkimus keskittyy ennemmin aineiston edustavuuteen kuin suuruuteen, minkä takia aineisto voi olla kooltaan suppea, kunhan se edustaa tutkittavaa ilmiötä hyvin. Laadullinen tutkimus tyypillisesti painottaakin yksittäisiä havaintoja numeerisen datan sijaan, minkä takia tartuinkin Eskolan ja Suorannan neuvoon valita niin pieni aineisto kuin mahdollista, mutta joka kuitenkin vastaa tutkimuskysymyksiin. (Eskola ja Suoranta 1998, 61–63.) Keskusteltuani heuristiikkojen luomiseen osallistuneen asiantuntijan kanssa päätin koota 80 topiikin suuruisen aineiston, jossa komponenttien topiikkijakauma huomioitaisiin, mutta kustakin komponentista olisi vähintään 10 topiikkia. Mikäli kylläänty mispistettä ei saavutettaisi 80 topiikilla, voisin kasvattaa aineistoa tarpeen mukaan. Loppujen lopuksi 80 topiikkia oli riittävä koko aineistolle.

Valittu aineistonkeruumenetelmä oli satunnainen poiminta, sillä silloin jokaisella topiikilla on yhtä suuri todennäköisyys tulla valikoiduksi, mikä puolestaan minimoi tutkijan subjektiiviset ennakkoasenteet ainestoa kohtaan. Poiminnan ohella varmistin, että kukin valituksi tullut topiikki oli todella julkaistu vuoden 2021 tuotejulkaisuissa tarkistamalla, että ne oli sisällytetty johonkin komponenttinsa DITA-mapeista. Käytin tähän tarkoitukseen oXygenin automaattista toimintoa, joka ei kuitenkaan paljastanut topiikkien sijaintia DITA-mapeissa.

5 Heuristisen arvioinnin keskeiset tulokset

Aineistossa korostui, miten tietyt ongelmat toistuivat kunkin komponentin dokumentaatiossa, mutta niissä kohdattujen haasteiden välillä oli myös eroja. Kaikki 19 heuristiikkaa kuitenkin tuottivat myös negatiivisia vastauksia, mikä osoittaa kunkin niistä olleen aineiston kannalta olennainen. Kuvaaja 1 näyttää kaikkien heuristiikkojen tulokset.

Kuvaaja 1. Kaikkien heuristiikkojen tulokset.



	H1.1	H2.1	H2.2	H2.3	H2.4	H3.1	H3.2	H3.3	H4.1	H4.2	H5.1	H5.2	H5.3	H5.4	H6.1	H6.2	H7.1	H7.2	H8.1
kyllä	56	74	45	47	53	18	22	37	45	63	26	22	9	64	49	35	36	49	26
ei	24	6	35	30	27	60	47	33	35	17	35	42	8	12	31	45	10	31	5
N/A	0	0	0	3	0	2	11	10	0	0	19	16	63	4	0	0	34	0	49

Topiikkien tarkastelu heuristiikan 1.1 avulla osoitti, että monen topiikin tyyppi oli väärä – yhteensä 24 topiikin olisi kuulunut käyttää toista topiikkityyppiä, ja osa tulisi jakaa useampaan eri topiikkityyppiin. Tyypillinen esimerkki tästä rikkomuksesta oli konsepti, jossa numeroitu lista <ol> (”ordered list”) toimitti tehtävissä käytettävän <steps> (”vaiheet”) elementin virkaa, tai sitten topiikissa sekoittui useampi informaatiotyyppi. Jälkimmäisessä tapauksessa myöskään modulaarisuuden periaate ei toteutunut. Koska topiikkityyppi oli valittu väärin, semanttisia elementtejäkin oli käytetty väärin, sillä konsepteilla ja tehtävillä on hyvin erilaiset DTD:t, eikä proseduurien edellyttämiä elementtejä voi käyttää konsepteissa.

Topiikkityypin valintaan liittyy myös topiikin otsikon muotoilu, sillä yksikön ohjeistukset määrittävät, että konseptien otsikot ovat substantiivilausekkeita, kun taas tehtävien otsikot alkavat pääverbillä, joka on gerundimuodossa. Yhteensä 27 topiikissa muotoilu oli toteutettu väärin, mikä saattaa hämmentää loppukäyttäjää, joka olettaa tietynlaisen otsikon nähdessään topiikin käsittelevän tietyntyyppistä informaatiota. Toisaalta otsikot saattoivat olla myös liian epämääräisiä esimerkiksi siitä syystä, että ne olivat vain yhden sanan mittaisia. Yksi sana ei

tyypillisesti riitä kuvaamaan monimutkaista konseptia tai monivaiheista tehtävää, etenkään jos otsikoksi valikoitunut sana on esimerkiksi ”Yleiskatsaus” (”Overview”).

Informaatiotyypittelyn ja topiikkien otsikoinnin ohella semanttisten elementtien käyttö oli myös merkittävä tarkastelunkohde. DITA-DTD:t ovat tarkoituksellisesti jäykkiä rakenteiltaan, koska niiden syntaksin on tarkoitus pakottaa kirjoittamaan sellaisella tavalla, joka tuottaa ennakoitavaa, rakenteellisesti yhtenäistä sisältöä (Stevens 2018). Väärän topiikkityypin valinta tietenkin aiheuttaa elementtien väärinkäyttöä, mutta sen lisäksi analyysissa korostui, että spesifien elementtien kuten <stepxmp> (”vaiheen esimerkki”) sijaan on käytetty yleisluontoisempia elementtejä, kuten <info>, vaikka elementin sisältämän tekstin puolesta olisi ollut tarkoituksenmukaisempaa käyttää jotakin muuta elementtiä. Hieman alle 60 % topiikeista käytti semanttisia sisäelementtejä (*inline elements*) väärin, kun taas lohkoelementtejä (*block elements*) käytettiin väärin jopa 75 %:ssa topiikeista.

Elementtien väärinkäyttö liittyi oleellisesti DITAn korostuselementteihin (*highlighting elements*) <b>, <i>, <u> ja <tt>, jotka poikkeavat semanttisista elementeistä siten, että ne vaikuttavat sisällön ulkonäköön. Näitä elementtejä ei kuitenkaan tule käyttää, jos sopiva elementti on saatavilla. Stevensin (2018) mukaan sisällön ulkonäön korostamisen sijaan tulisi kuitenkin miettiä, mitä kirjoitetaan ja miksi – tästä syystä DITA ei sisällä informaation korostamiseen varattua elementtiä. Etenkin tekstin lihavointiin käytettävää elementtiä <b> käytettiin aineistossa, sillä se esiintyi yhteensä 27 topiikissa, kun taas elementtiä <i> käytettiin 8 topiikissa. Näiden elementtien käyttötavat vaihtelivat komponenttien välillä, toisinaan jopa yksittäisten topiikkien sisällä, mikä kertonee siitä, että niiden käyttö on jollain tapaa tiedostamatonta. Aineiston perusteella etenkin korostuselementtiä <b> saatetaan käyttää siitä syystä, että tiettyjen elementtien kuten <uicontrol> (”user interface control”) tiedetään olevan lihavoituja myös dokumentaation julkaisumuodoissa. Tämä ei ole kuitenkaan hyväksyttävä syy olla käyttämättä semanttista merkintää. Lisäksi lähdeformaattiin jää tällöin ulkonäköön viittaavaa kontekstia, mikä ei DITA-ympäristössä ole toivottavaa.

Uudelleenkäyttö on ehkäpä suurin DITAn tarjoama hyöty, ja sitä hyödynnettiin parhaiten komponentissa B, jossa variaabeleiden käytön lisäksi uudelleenkäytettiin myös sellaisia tehtävien vaiheita, jotka toistuvat useassa topiikissa. Variaabeleiden käyttö sen sijaan oli yhdenmukaista ainoastaan komponentissa D – muissa komponenteissa variaabeleiksi sopivia tai jo sellaisiksi määriteltyjä erisnimiä olikin toisinaan niin sanotusti ”kovakoodattu” eli kirjoitettu tägäämättömänä tekstinä. Tutkimusaineiston heuristisen arvioinnin perusteella myös

muiden uudelleenkäytettävien sisältöjen, kuten topiikkien sisäisten tai välisten viittausten, kovakoodaaminen alentaa dokumentaation uudelleenkäytettävyyspotentiaalia.

Oli yllättävää huomata, että yhteensä yli 60 % topiikista ei sisältänyt ainuttakaan linkkiä eli DITAn sisäistä viittausta. DITA-ympäristössä on tarkoitus luoda ”informaation verkko” luomalla linkkejä DITA-topiikkien välille eri keinoin (Bellamy et al. 2011, luku 1). Esimerkiksi tehtävissä voisi olla linkkejä konsepteihin tai referensseihin, jotka liittyvät oleellisesti tehtävissä kuvattuihin ohjeisiin. Linkkien tarkoituksena on tukea modulaarisuutta ja käyttäjäystävällisyyttä, kun kuvailevaa informaatiota ei tarvitse sisällyttää itse tehtävään, vaan tehtävän lukijalle voi antaa mahdollisuuden siirtyä tarkastelemaan konseptia, mikäli hän kokee sen tarpeelliseksi. Tähän tarkoitukseen on oma elementtinsäkin, <related-links> (”aiheeseen liittyvät linkit”), mutta sen käyttöä ei tarkasteltu arvioinnissa, sillä sitä ei ole erikseen ohjeistettu käyttämään yksikössä. Toisaalta linkit ja viittaukset ei-tarpeellisiin sisältöihin rikkovat niin minimalismin kuin yksikönkin ohjeita vastaan. Jos olisin tutkinut DITA-mappeja, olisin voinut selvittää, puuttuiko topiikeista tarpeellisia linkkejä.

Myös minimalismin ja tehtävien korostamisen teemat korostuivat analyysissä. Noin 40 %:ssa topiikeista esiintyi toisteista tai itsestäänselvää informaatiota, mikä heikentää dokumentaation luettavuutta ja saattaa turhauttaa käyttäjää, joka haluaisi päästä tositoimiin sen pidemmittä puheitta. Carrollin ja van der Meij’n (1995, 244) mukaan minimalistinen ohjeistus rohkaisee käyttäjää kokeilemaan ja oppimaan itse sen sijaan, että ohjeet annetaan valmiiksi pureskeltuina. Tällöin tehtävää suorittava käyttäjä tuskin tarvitsee <stepresult> (”vaiheen tulos”) elementtiä, joka ilmoittaa nettisivun aukeavan, kun käyttäjä on tehtävän ohjeistuksen mukaisesti kirjoittanut sivun osoitteen selaimeen ja painanut Enter-näppäintä (ks. esimerkki 11, luku 5.7).

Neljän komponentin 80 topiikin arvioinnin tuloksissa oli siis sekä vastaavuuksia että eroavaisuuksia. Osa havaituista ongelmista oli muita vähäisempiä, mikä tarkoittaa, että niihin vaikuttaminen lienee mahdollista verrattain pienellä vaivannäöllä. Variaabelit ovat tällainen esimerkki, ja komponentin C dokumentointitiimi onkin hiljattain ottanut variaabelit käyttöön. Komponenttien väliset tulokset eivät kuitenkaan eronneet toisistaan merkittävästi, jolloin näkisin mahdollisena tiimien välisen yhteistyön havaittujen puutteiden korjaamiseksi.

6 Päätelmät

Tutkimustehtävänäni oli selvittää, miten DITA XML:ää on käytetty kohdeyrityksen yhden liiketoimintayksikön neljän eri komponentin asiakasdokumentaatiossa. Dokumentaatiota

tarkastelemalla tutkittavia teemoja olivat DITAn periaatteiden lisäksi rakenteinen kirjoittaminen, minimalismi ja yksikön omat sisäiset ohjeistukset. Tutkimusaineisto kattoi 80 DITA-topiikkia, joista puolet oli konsepteja ja puolet tehtäviä. Valittu tutkimusmenetelmä oli heuristinen arviointi, joka edellytti heuristiikkalistan luomista. Lopputuloksena on 19 DITA-heuristiikan lista, jonka laatimiseen ja onnistuneisuuteen vaikutti suuresti yksikön dokumentaatioasiantuntija.

Vaikkakin dokumentaation sisällössä oli suurta vaihtelevuutta, tulosten perusteella tietyt piirteet korostuvat kaikissa komponenteissa. Tutkimuksen johtopäätös onkin, että DITAlla laadittu dokumentaatio ei täysin vastaa DITAn, minimalismin tai rakenteisen kirjoittamisen periaatteita, eikä se myöskään noudata yksikön sisäisiä ohjeistuksia kaikilta osin. Mahdollinen selitys tälle on se, että vaikka DITAa voikin käyttää lukemattomilla tavoilla hyödykseen, valittuja tapoja tulisi olla vain yksi kerrallaan, tai muutoin lopputuloksena on dokumentaation epäjohdonmukaisuutta ja hankaluuksia DITAn tarjoamien hyötyjen saavuttamisessa. Tämä päätelmä oli kuitenkin odotettavissa, sillä dokumentaatiota laaditaan rajallisten resurssien varassa, ja tuotekehitykseltä saatava materiaali saattaa olla puutteellista tai tulla viime tingassa, jolloin laatu kärsii aiheutuneen aikapaineen takia. Jatkotutkimuksen aiheena olisikin kiinnostavaa selvittää, mitkä tekijät aiheuttavat ristiriidan laaditun dokumentaation todellisuuden ja teorian välillä. Loppujen lopuksi käyttäjät määrittelevät dokumentaation laadun (Virtaluoto et al. 2021, 32). Toisaalta kuten kohdeyrityksen asiantuntija toteaa, dokumentaatio on toisinaan tuotteen peili – monimutkainen tuote saattaa tuottaa monimutkaista dokumentaatiota, eikä tiettyjä puutteita voida välttämättä kompensoida dokumentaatiossa.

Heuristinen arviointi oli soveltuva menetelmä tutkimuskysymysten käsittelemiseen. Tutkimus olisi kuitenkin voinut hyötyä siitä, jos arvioijia olisi ollut useampi, tai jos havainnoista olisi ollut mahdollista käydä syväluotaavaa keskustelua ryhmässä. Koska heuristiikat laadittiin yksittäisten topiikkien eikä kokonaisten DITA-mappien tai dokumenttien arvioimiseen, tutkimustulokset eivät ole yleistettävissä topiikkeja laajempiin kokonaisuuksiin. Tulokset tuovat kuitenkin hyödyllistä tietoa siitä, miten DITAa käytetään ja millaisia kysymyksiä sen käyttöön liittyy kohdeyrityksessä. Tuloksia tullaankin käyttämään yksikössä osana transformaatioprojektia, minkä lisäksi heuristiikkalistaa laajennetaan työkaluksi kirjoittajille. Tässä tutkimuksessa DITAn käyttöä arvioi yksi kirjoittaja, mutta olisi kiinnostavaa tutkia aihetta teknisten kirjoittajien näkökulmasta käsin – miten he näkevät DITAn käytön?