VALUE OF INTERNAL TOOL DEVELOPMENT TO SOFTWARE DELIVERY PROJECTS

Lappeenranta–Lahti University of Technology LUT

Degree Programme in Industrial Engineering and Management, Master's Thesis

2021

Janne Krutsin

Examiners: Associate professor Kalle Elfvengren

Professor Marko Torkkeli

ABSTRACT

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT School of Engineering Science
Degree Programme in Industrial Engineering and Management

Janne Krutsin

Value of Internal Tool Development to Software Delivery Projects

Master's Thesis
2021
63 pages, 8 figures, and 9 tables
Examiners: Associate professor Kalle Elfvengren and Professor Marko Torkkeli

Keywords: internal tool, internal product, innovation, software

Despite the advancements in technologies and overall processing power, many real-world tasks are still bottlenecked by the lack of appropriate tooling. This lack of tooling leads to more manual work and slower project progress. However, some tools can be created by software delivery projects for specific use cases to achieve better efficiency. This thesis focuses on software-based internal tools in delivery projects. The objectives are to define the term internal tools, examine the prerequisites and effort required for their development, and investigate their value to software delivery projects.

Software-based internal tools can be defined as auxiliary tools developed by employees inside an organization for specific internal use cases without productization purposes, internal or external. Unmodified third-party tools and open-source projects, simple derivatives of third-party tools or open-source projects, and software languages, libraries, frameworks, or protocols are not internal tools. Internal tools are innovations, and as such, need to be functional and implementable. Included case studies revealed that the benefits of internal tools are usually derived from repetitive use cases or use cases that enable the delivery project. The existence of these benefits is also a prerequisite for internal tool development. The tool's scope will determine the effort its development requires, which is restricted by the delivery project's limits.

The resources an internal tool saves or generates for the delivery project will determine its value. Value can be either calculated from known variables or estimated. Estimating the internal tool's value is done by comparing its benefits to the effort used for the development. This report proposes the internal tool value estimation matrix to help with the value estimation of an internal tool. The proposed matrix categorizes internal tools into high- and low-value tools in addition to a middle section that indicates possible problems with the tool.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUT School of Engineering Science
Tuotantotalouden koulutusohjelma

Janne Krutsin

Sisäisten työkalujen kehityksen arvo ohjelmistotoimitusprojekteille

Diplomityö
2021
63 sivua, 8 kuvaa ja 9 taulukkoa
Tarkastajat: tutkijaopettaja Kalle Elfvengren ja professori Marko Torkkeli

Hakusanat: sisäinen työkalu, sisäinen tuote, innovaatio, ohjelmisto

Teknologian ja laskentatehon kehityksistä huolimatta useita reaalimaailman tehtäviä rajoittaa edelleen sopivien työkalujen puuttuminen. Työkalujen puuttuminen lisää manuaalisen työn määrää ja hidastaa projekteja. Joidenkin käyttötapauskohtaisten työkalujen luominen onnistuu kuitenkin projektitiimin jäsenien toimesta ohjelmisto-toimitusprojektin toiminnan tehostamiseksi. Tämä työ keskittyy ohjelmistopohjaisten sisäisten työkalujen kehittämiseen ohjelmistotoimitusprojekteissa. Työn tarkoituksena on määritellä termi sisäiset työkalut, tarkastella edellytyksiä ja vaadittavaa työmäärää sisäisten työkalujen kehitykselle sekä tutkia niiden arvoa ohjelmistotoimitusprojekteille.

Ohjelmistopohjaiset sisäiset työkalut voidaan määritellä avustaviksi työkaluiksi, joita kehitetään organisaation sisäisten työntekijöiden toimesta erityistä käyttötarkoitusta varten. Sisäisiä työkaluja ei kehitetä ulkoisiksi eikä sisäisiksi tuotteiksi. Pelkät muokkaamattomat kolmannen osapuolen kehittämät työkalut, avoimen lähdekoodin projektit, yksinkertaiset kolmannen osapuolen työkalujen tai avointen lähdekoodiprojektien johdannaiset, ohjelmointikielet, koodikirjastot, viitekehykset tai protokollat eivät ole sisäisiä työkaluja. Sisäiset työkalut ovat innovaatioita ja siten niiden on oltava sekä implementoitavissa että funktionaalisia. Työn tapaustutkimukset osoittavat, että sisäisen työkalun hyödyt voidaan yleensä johtaa toistuvista käyttötapauksista tai käyttötapauksista, jotka mahdollistavat ohjelmistotoimitusprojektin. Käyttötapausten hyödyllisyys on myös sisäisten työkalujen kehityksen edellytys. Työkalun toiminta-ala määrää sen kehitykseen vaadittavat resurssit. Näiden resurssien määrää rajoittavat toimitusprojektin asettamat rajoitukset.

Sisäisen työkalun säästämät ja tuottamat resurssit määrittävät sen arvon. Työkalun arvo voidaan laskea tunnettujen muuttujien perusteella tai arvioida. Työkalun arvoa arvioidaan vertaamalla sen hyötyjä kehitystyön vaatimiin resursseihin. Tämä työ esittelee sisäisten työkalujen arvon arviointimatriisin helpottamaan arvon määrittelyä. Esitelty matriisi kategorisoi sisäiset työkalut projektin kannalta arvokkaisiin, vähäarvoisiin ja mahdollisia ongelmia sisältäviin työkaluihin.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my manager Matti Koivula for going the extra mile to enable me to complete this thesis alongside my day job. I would also like to extend my sincere thanks to my senior manager Ville Tuulos for his practical suggestions and guidance for this thesis. Special thanks to my co-workers David Lee for reviewing this thesis and Seppo Kallio and Ville Linnalehto for providing crucial information for the case studies.

Thanks should also go to the examiners, Associate Professor Kalle Elfvengren and Professor Marko Torkkeli at LUT University, for their patience which cannot be underestimated.

I am also grateful to my friends Eeli Asukka and Sami Uronen, who pushed me forward at times when I was losing momentum. I will never forget the support, care, and constructive criticism received in our Telegram group chat DippaT.

Finally, my deepest gratitude goes to my family and my significant other, Henna Janatuinen, for supporting me through the whole process. This accomplishment indeed would not have been possible without them. Thank you.

6.12.2021 Lappeenranta

Janne Krutsin

LIST OF ABBREVIATIONS

API	Application Programming Interface
CC	Case Company
CCS	ECM solution by the Case Company
ECM	Enterprise Content Management
ICT	Information and Communications Technologies
MVP	Minimum Viable Product
OECD	Organization for Economic Co-operation and Development
SaaS	Software as a Service
UI	User Interface

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This introductory chapter lays the groundwork for the study by glancing at the factors that make it topical and outlines its technical aspects. The research questions defined in this chapter will summarize the agenda for the study. Importantly, the delimitations set the scope for the study, making an essential separation between concepts that usually are used as synonyms.

1.1 Research background

While the raw power of technology has risen dramatically for decades, the symbiotic relationship between computers and humans has not been able to keep up. Computers and especially cloud computing have provided great leaps in processing power. Still, the person controlling the computer resources is the bottleneck for completing many tasks faster and more efficiently in many real-world applications. This inefficiency can be due to multiple reasons, e.g., manual steps in the process, too generalized tools, or a lack of effective tooling. Until algorithms can generate new algorithms themselves for real-world use cases, many real advancements in task completion are obtained with custom-built software and a more intuitive link between the user and the computer.

Even if everything cannot be optimized fully, completing real-world tasks or even whole processes has been substituted with software solutions that have enough demand on the market. For tasks too specific to be found broadly from certain fields, markets, or even a single organization, well-developed commercialized solutions or internal bureaucratic products do not have answers. However, the fact that a niche solution is absent from general markets does not mean an absence of demand for this solution. Internal tools answer this demand by allowing tasks to be transferred to computers.

The advantages of increased efficiency in organizations are numerous (Ling, Gautam, and Vallabh, 2012, pp. 520–525), and these advantages have a noticeable effect, especially on incurred labor-related expenses. Labor costs have been steadily rising in Europe (Eurostat, 2021), and consequently even minor efficiency enhancements can significantly impact the organization's finances. More importantly, the time savings dimension as a whole has grown in importance, especially in the ICT sector during the past decade. In the era of SaaS or Software as a Service in software companies (Gartner, 2021a; Wohl and Simon, 2010, p. 98), not only are inefficiently spent work hours expensive as a labor cost, but slow delivery projects usually lead directly to missed revenue due to lost subscription payments (Wohl *et al.*, 2010, pp. 107–111). As software companies stand to benefit from more efficient work processes and many ITC sector experts have prior knowledge or experience in enhancing their work processes with IT solutions, software delivery projects are excellent subjects for studying the benefits of internal tools.

The topic of internal tools is not interesting only because of its importance; it also seems to be a topic that has not been widely studied. Examining academic sources while working on this thesis revealed that most of the mentions of internal tools seem to refer to something other than internally used tools. This topic could have been on many researchers' minds already, but it has been simply passed as an example or trait to a higher level innovation concept.

Still, the actual topic of this study is firmly rooted in the experiences faced every day by the researcher and many others working in the ITC sector. Even though there seems to be little to no academic studies recognizing non-productized tools as a variable for an organizations' success, these purpose-built, highly specified tools are everywhere.

1.2 Research objective, questions, and methodology

The main objective of this research is to explore the value of internal tool development to software delivery projects. This main objective requires other questions to be answered before the main goal is reached. As internal tools have not had much attention from the research community, the first research question aims to answer what internal tools are and how they are connected to innovation. The second research question explores the prerequisites for internal tool development and what type of effort the development takes. These essential questions allow us to understand the core concepts better and to answer the third and main research question, whether internal tool development brings value to software delivery projects and how that value can be measured. The research questions are:

RQ1: What are internal tools in software delivery projects?

RQ2: What are the prerequisites for internal tool development, and what type of effort does their development take?

RQ3: Does internal tool development bring value to software delivery projects, and how can that value be measured?

The study utilizes a literature review for answering the first and the second research questions. As a primary method, a case study methodology is used to explore real-world examples of the topic at hand. Two qualitative case studies uncover answers to the second and the third research questions. The information for the case studies have been gathered from project documentation, interviews and reports. Together these methods will provide a solid base for the report's conclusions.

1.3 Delimitations

This study is limited to software-based tools that are built inside an organization for a specific internal need. This means that the purpose for developing the tools will act as the primary limiting factor for deciding if the tool can be considered to fit the scope of this study. Tools intended for this kind of internal use are not developed with productization in mind. However, these internal tools can be productized later in their lifecycle when matured beyond their original use cases. A *use case* describes how the software will be used (Pressman, 2001, p. 280). Even if the use cases do not give a definite indication whether the tool is developed with productization in mind, use cases that grow in a more general direction can be evidence of a tool which is outside the scope of an internal tool.

The definition of productization includes productizing internal tools for both internal and external use cases. While internal products will serve the same purpose as internal tools studied in this work, internal products usually have predefined resources and might even have assigned persons for their development. Internal tools in the context of this study are tools that need to be developed more quickly and will serve primarily one purpose at the start. Internal tools and their relationship with internal products are defined in more detail in chapter 2. Later in this work, a general term *auxiliary tool* will describe tools used internally.

This study will also focus on innovating these tools and their benefits instead of the actual software development processes involved in the development of these tools. While there might be significant advantages to the development speed and the actual usefulness of the tool when using the best theoretical models to guide the development, internal tools might not be developed only by seasoned developers with an extensive understanding of software development. For this reason, technical differences between different approaches in theoretical software development practices are not included in this study.

Some technical limitations on different technologies are still in order. Even if adopting new technology can yield significant benefits and could be viewed as a new tool for organizations, this study focuses on the internal innovation processes. This study will not recognize already productized third-party-developed tools or open-source projects nor their simple derivatives as internal tools. Also, plain frameworks, software languages, or protocols are outside of the scope of this study.

Also, even though better tooling can, in general, provide value to many different kinds of projects, this work focuses on software delivery projects. Software *delivery* or *deployment* includes delivering software either in parts or in one entity (Pressman and Maxim, 2015, p. 17). Hence, software delivery projects are projects that deliver software to customers.

1.4 Structure of the study

The structure of this study consist of theoretical and empirical parts. The theoretical part starts with the literature review and attempts to answer the first research question; what are internal tools in software delivery projects? This literature review will also provide background knowledge about innovation in later chapters. The empirical part of the study accompanies answering the second research question. The empirical part consists of two qualitative case studies. These case studies offer an insight into the real-world process of internal tool development and provide answers to the questions; what are the prerequisites for internal tool development and what type of effort is required. Finally, after understanding more about the nature of internal tools, we can focus on answering the third and main research question; does internal tool development bring value to software delivery projects, and how can that value be measured. The case studies provide the answer to this question. The structure of the study is laid out in Figure 1 with short summaries of the results.

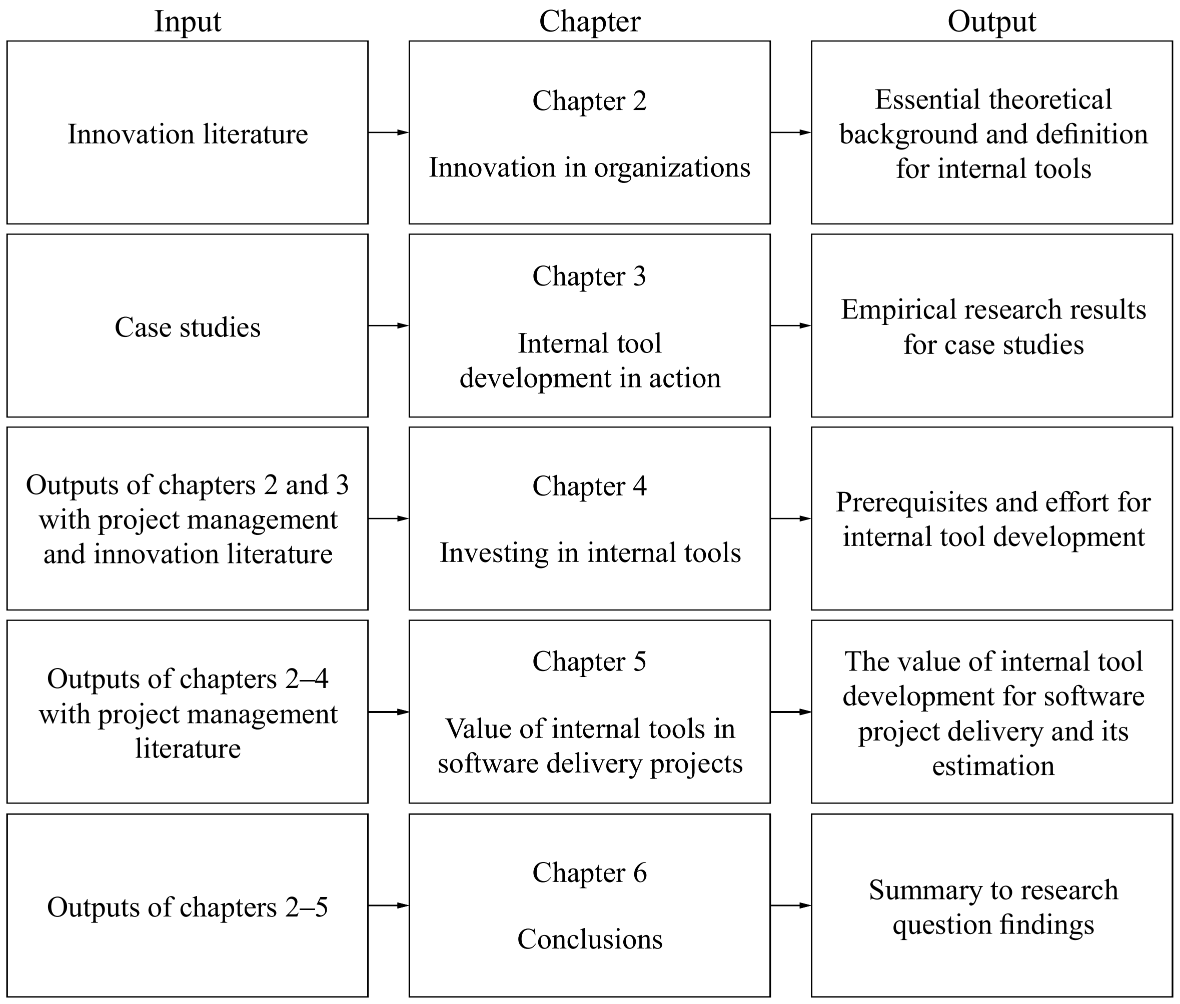
Input
Chapter
Output
Innovation literature
Chapter 2
Innovation in organizations
Essential theoretical background and definition for internal tools
Case studies
Chapter 3
Internal tool development in action
Empirical research results for case studies
Outputs of chapters 2 and 3 with project management and innovation literature
Chapter 4
Investing in internal tools
Prerequisites and effort for internal tool development
Outputs of chapters 2–4 with project management literature
Chapter 5
Value of internal tools in software delivery projects
The value of internal tool development for software project delivery and its estimation
Outputs of chapters 2–5
Chapter 6
Conclusions
Summary to research question findings

Figure 1. Input-output chart presenting the structure of the report.

Lastly, the discussions and conclusions chapters will summarize the findings for the research questions and the main objective for the study. This chapter will also highlight the new knowledge claims presented in previous chapters. Discussion topics that emerge from the key findings are offered in the last part of the chapter.

2 INNOVATION IN ORGANIZATIONS

This chapter is the theoretical part of the study and is divided into five sections. The first three sections will lay the theoretical groundwork for the study by exploring innovation core concepts and categorizations. The fourth section is a literature review of internal auxiliary tools. After exploring specified literature, the term internal tool is defined in section five and placed into the previously explored theoretical categorizations.

2.1 Innovation or creativity?

New tools, whether they are for internal or external use cases, can originate from creative ideas. A person discovers a new way to get the task done. What separates creativity from innovation, and are all new tools just a result of a person's creativity?

There are multiple definitions for creativity and innovation. Amabile *et al.* (1996, p. 1155) define *creativity* as: "the production of novel and useful ideas in any domain" and *innovation* as: "the successful implementation of creative ideas within an organization." While these simple definitions do not describe creativity or innovation completely, they are clear about the author's point of view. In turn, Hughes *et al.* (2018, pp. 8–9) present an updated, more complex, and complete recommendation for definitions of *creativity* and *innovation* in their article. According to them:

> Workplace creativity concerns the cognitive and behavioural processes applied when attempting to generate novel ideas. Workplace innovation concerns the processes applied when attempting to implement new ideas. Specifically, innovation involves some combination of problem/opportunity identification, the introduction, adoption or modification of new ideas germane to organizational needs, the promotion of these ideas, and the practical implementation of these ideas.

Both of these definition pairs put forward idea generation as the main feature of creativity. Similarly, both definition pairs have idea promotion and idea implementation as main features for innovation. Hughes *et al.* (2018, pp. 8–9) have specifically mentioned idea promotion, and while Amabile *et al.* (1993, p. 1155) do not, they infer it by defining that the implementation of ideas needs to be successful. In addition to these usual features, Hughes *et al.* (2018, pp. 8–9) have recognized other distinguishing features for creativity and innovation. According to the study, while creativity has novelty as a requirement, innovation does not. Innovation can also be an enhancement to a previously existing idea. In turn, only innovations have a functional requirement as they are created to improve organizational outcomes. Innovations are also more concrete as creativity takes place mainly in a person's cognition, while innovation presents itself in interpersonal, social, and practical contexts. The results of creativity and innovation mirror previously recognized usual features by stating that the result of creativity is an idea, which is a result of idea generation. In contrast, the result for innovation is a functional and implemented idea, which is, in turn, something implemented. These features are collected in Table 1 below.

Table 1. Features distinguishing creativity and innovation.

Feature	Creativity	Innovation
Idea generation	Yes	No
Idea promotion	No	Yes
Idea implementation	No	Yes
Novelty requirement	Yes	No
Functional requirement	No	Yes
Where does it take place?	Cognition	Interpersonal, social, and practical context
Result	An idea	A functioning and implemented idea

Note. Adapted from "Leadership, creativity, and innovation: A critical review and practical recommendations" by Hughes, Lee, Tian, Newman, and Legood, 2018, The Leadership quarterly, 29, pp. 81.

When a person has an idea about a tool, it is just that, an idea. However, when this idea is promoted and implemented, it becomes an innovation. The same core idea can be found in Brown and Katz's (2011, p. 381) model of the three spaces of innovation. According to them, people move through the three spaces of innovation while creating innovations. These spaces are inspiration, iideation, and implementation. The *Inspiration space* is the motivation for searching for new solutions. *Ideation*, on the other hand, is where the idea is developed.

Implementation is the final space where an idea leaves the project room, making it an innovation. As this study focuses on tools created inside an organization, the process must be explicitly analyzed from the point of innovation.

To incorporate more structure for this literature review and to better answer the question; how internal tools are developed, the attributes of innovation need to be considered. Baregheh, Rowley, and Sambrook (2009, pp. 1331–1332) have divided the attributes of innovation into nature of innovation, type of innovation, stages of innovation, social context, means of innovation, and aim of innovation. With the *nature of innovation*, they refer to the novelty of the innovation. Innovations can be either completely new and novel or an enhancement of an existing innovation. *Type of innovation* describes the type of output or result of an innovation. While type refers to the result, *stages of innovation* refer to all of the steps in the actual innovation process. On the other hand, *social context* refers to the subjects that affect the innovation process. These can either be people involved in the innovation process, environmental factors, or other social entities. *Means of innovation* describe the resources needed for the innovation to be completed, and the *aim of innovation* refers to the desired result of innovation. Table 2 summarizes these attributes from Baregheh *et al.* (2009) with short descriptions.

Table 2. Attributes of innovation.

Attribute	Description
Nature of innovation	Form of innovation
Type of innovation	Type of output
Stages of innovation	Innovation process steps
Social context	Social entity, system or group involved in innovation process
Means of innovation	Necessary resources for innovation process
Aim of innovation	Overall result expected

Note. Adapted from "Towards a multidisciplinary definition of innovation" by Baregheh, Rowley and Sambrook, 2009, Management decision, 47, pp. 1331–1332.

The nature, means, and aim of innovation is relatively easy to understand in the context of this work. However, the type, stages, and social context of innovation needs more background before delving into the practical side of the report. These attributes are examined further in sections 2.2 and 2.3.

2.2 Organizational culture and innovation

The last section concluded that an auxiliary tool is an innovation, an implemented idea. However, innovation can also be viewed as a process. According to Baregheh *et al.* (2009, pp. 1331–1334), the multi-staged process of promoting and implementing ideas is also an innovation. This definition means that innovation is, at the same time, the process itself and also the product of this process. This process is influenced by countless different variables, many of which have been studied in the literature (Baregheh *et al.*, 2009). However, this study focuses on tools not developed as a generic product but for a specific internal need. As the employees face these particular needs which cause new ideas to form, the innovation process for auxiliary tools depends heavily on the individuals' willingness to promote their new idea. After all, *organizational culture* can be described as behavioral patterns caused by the surrounding environment (Lewis and Kaiser, 2019, p.15). This fact highlights the social context of innovation. Let us look at some cultural values that affect employees' willingness to advance with the innovation process.

Innovation is many times brought up in the context of an organization's offerings to the market. According to Büschgens, Bausch, and Balkin (2013, pp. 769–777), this innovativeness in external offerings does not necessarily mean that the company's internal culture supports innovation. However, unlike products or services produced to the market, internal support for innovation is vital in the case of auxiliary tools, as will be recognized in section 2.5.

Büschgens *et al.* (2013, p. 764) found that at least 40 different cultural values have been linked to innovation. Some of those values are broader concepts, like innovation culture and supportive culture. They also identified many, much more specific cultural variables. It is important to note that these variables are not purely innovation fostering. According to Büschgens *et al.* (2013, p. 777), some studies also found aspects of cultures to inhibit innovation. It seems clear that these variables affect the organization's internal innovation capabilities. As auxiliary tools benefit from a culture that supports innovation, innovation culture as a concept needs to be defined further.

Dobni (2008, p. 540) found out in their article that the definition of innovativeness in an organization can vary widely from simple intention to be innovative to specific requirements for the innovation process and the result. They defined *innovation culture* as:

> multi-dimensional context which includes the intention to be innovative, the infrastructure to support innovation, operational level behaviors necessary to influence a market and value orientation, and the environment to implement innovation.

As success is usually measured by comparing changes in organizations' performance, they searched for innovation culture factors from previously mentioned dimensions. These dimensions are presented in Dobni's (2008, p. 541) model of innovation (Figure 2).

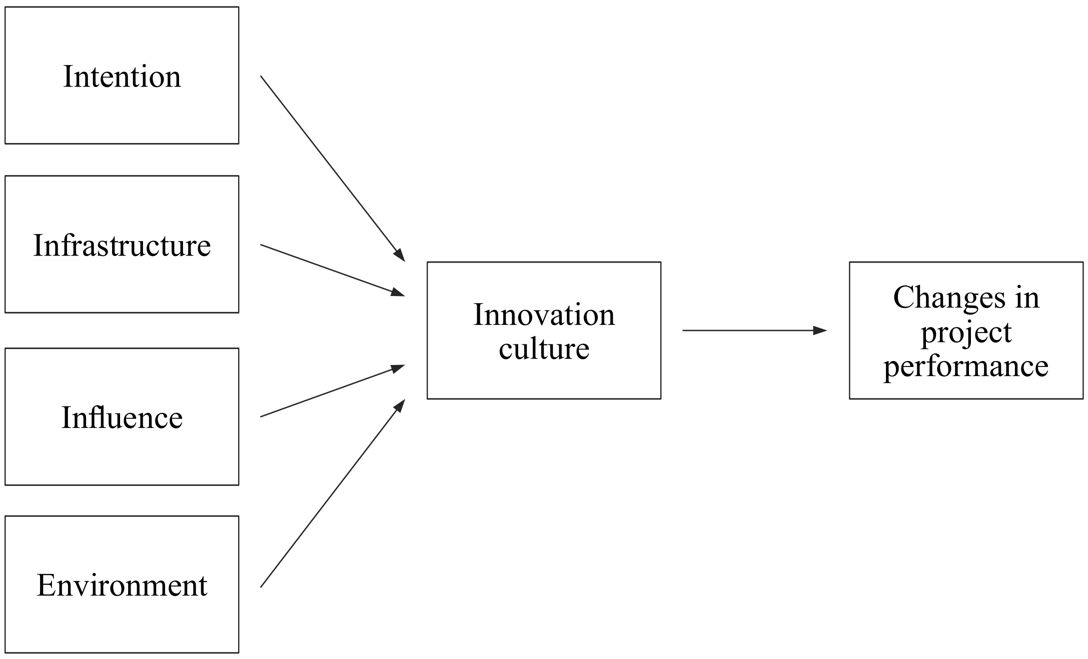

Figure 2. Dimensions of innovation culture.

Note. Adapted from "Measuring innovation culture in organizations: The development of a generalized innovation culture construct using exploratory factor analysis" by Dobni, 2008, European Journal of Innovation, 11(4), p. 541.

From these four dimensions, Dobni (2008, pp. 549–552) derived seven factors with a strong correlation that they found to represent innovation culture. These factors were innovation propensity, organizational constituency, organizational learning, creativity and empowerment, market orientation, value orientation, and implementation context. Dobni has considered these factors at the organizational level from a general standpoint. As the factors apply to innovations in general, examples can also be derived from auxiliary tool innovation.

Dobni (2008, pp. 549–552) has described *innovation propensity* as the extent to which an organization has a formalized — and integrated — infrastructure for developing and sustaining innovation. In the context of auxiliary tool development and adaption, this factor describes how well an organization has adopted auxiliary tools as a part of its processes. *Organizational constituency* was described by Dobni (2008, pp. 549–552) as the degree to which employees are committed to the innovation imperative and how individuals view themselves in relation to their coworkers regarding their value, equity, and contributions made within their company. Specifying this factor to the level of auxiliary tools considers the degree to which employees are engaged in the innovation imperative regarding auxiliary tools. The next factor, *organizational learning*, describes how education and training options match with the organization's innovation goals. This factor describes how well these training options support auxiliary tool development in the context of auxiliary tools. (Dobni, 2008, pp. 549–552.)

By *creativity and empowerment*, Dobni (2008, pp. 549–552) means how employees are allowed to be creative in their employment. It also evaluates employees' levels of empowerment as well as their ability to improvise and act independently. Furthermore, this factor includes both the limits set for employees' creativity and the creative potential within employees for auxiliary tools. Articles from Amabile (1983, p. 370) and Abbey and Dickson (1983, p. 366) support this factor. Amabile found support for causality between externally set limits for employees and hindered creativity. Abbey and Dickson, in turn, found that creative freedom regarding the experimentation of new ideas is a part of an innovative work climate.

Dobni (2008, pp. 549–552) links *market orientation* to employees' attributes. According to them, market orientation as a factor considers the degree to which employees develop and communicate information about customers, competitors, the industry, and their grasp of the value chain or cluster in which they work. Even when auxiliary tools do not have markets in the general sense but may have value for organization's internal processes, auxiliary tools are not valuable by themselves. In the context of auxiliary tools, market orientation describes an employee's knowledge of the value proposition to the end-user or client and how well auxiliary tool development supports it. *Value orientation* is also heavily linked to employees' properties. Dobni (2008, pp. 549–552) describes value orientation as the extent to which employees are concerned with and involved in generating value for end-users or clients. In the context of auxiliary tools, value orientation describes the degree of involvement in creating value for internal processes.

The last and maybe the most important factor is the *implementation context*. Dobni (2008, pp. 549–552) found that this had the most significant impact on performance changes in an organization. They described this factor as part of an organization's ability to put value-added ideas into action. In addition, Chandler, Keller, and Lyon (2000, pp. 67–73) came to similar conclusions in their article; the environment and culture need to fit together. This factor can be seen as the organization's ability to implement auxiliary tools. All seven factors are presented in Figure 3.

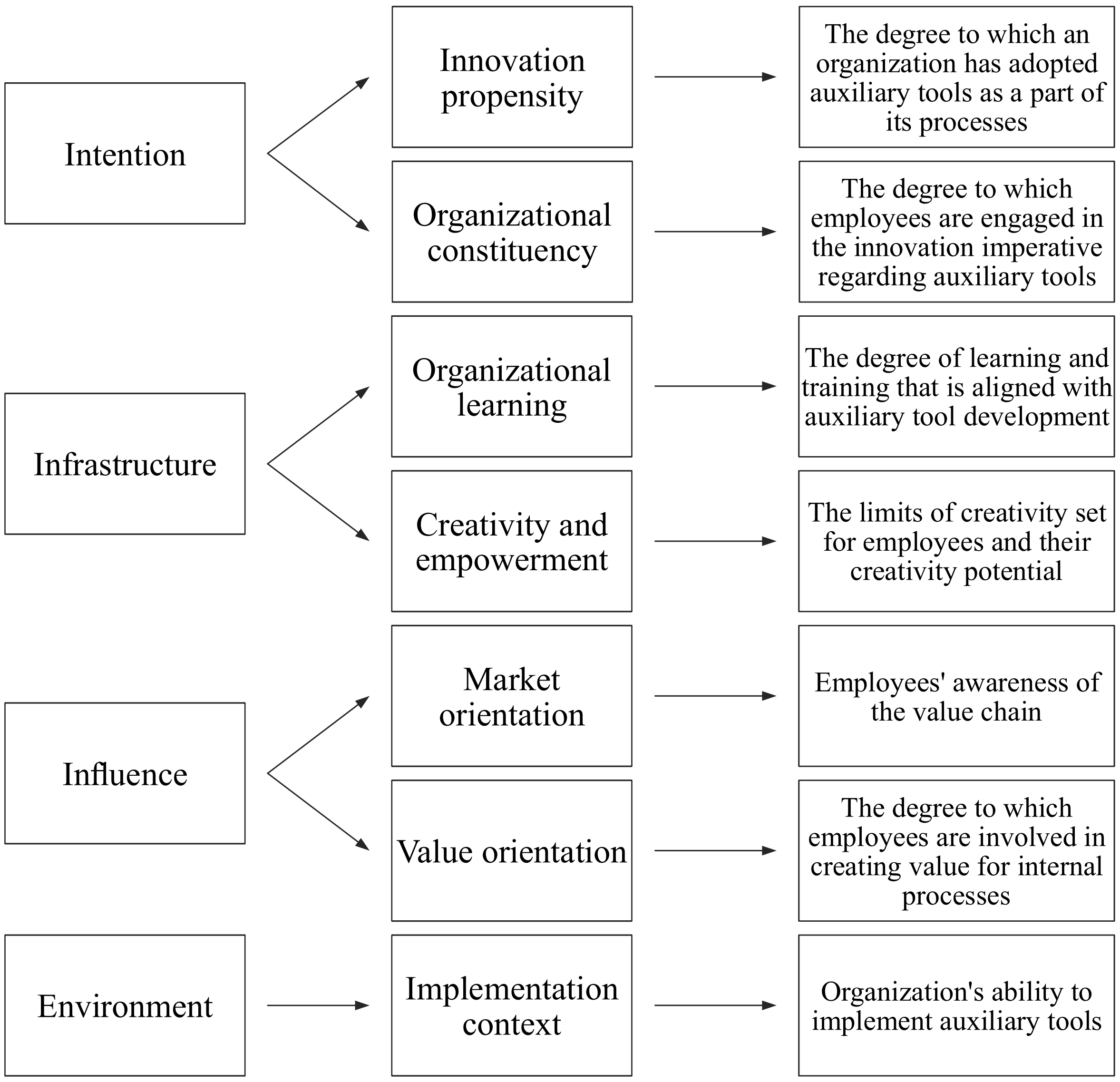
Intention
Innovation propensity
The degree to which an organization has adopted auxiliary tools as a part of its processes
Organizational constituency
The degree to which employees are engaged in the innovation imperative regarding auxiliary tools
Infrastructure
Organizational learning
The degree of learning and training that is aligned with auxiliary tool development
Creativity and empowerment
The limits of creativity set for employees and their creativity potential
Influence
Market orientation
Employees' awareness of the value chain
Value orientation
The degree to which employees are involved in creating value for internal processes
Environment
Implementation context
Organization's ability to implement auxiliary tools

Figure 3. Correlation factors for innovation culture regarding auxiliary tools.

Note. Adapted from "Measuring innovation culture in organizations: The development of a generalized innovation culture construct using exploratory factor analysis" by Dobni, 2008, European Journal of Innovation, 11(4), pp. 551.

These correlation factors are one example of the complexity of the social context of innovation. Thankfully, examining the dimensions of a broad concept such as innovation culture makes it easier to grasp. Even though Dobni (2008, pp. 546–552) found that some correlating factors have more weight than others, the importance can still change between different situations.

2.3 Types of innovation

According to Baregheh *et al.* (2009, p. 1331–1333), the type of innovation refers to the type of result or output of innovation. This viewpoint focuses on *what* but not on *how*. They classify innovations into four categories; product, service, process, and technical innovations. However, just as the definition of innovation has a significant variance in its content, so does the categorization of innovation types. For example, Damanpour (1996, p. 694) splits them into product, service, process and organizational/administrative innovations whereas Trott (2012, p. 17–18) splits innovations into seven categories; product, service, process, commercial/marketing, management, organizational, and production innovations. Many other categorizations that focus on different perspectives have not been mentioned in the previous listing. Baregheh *et al.* (2009, p. 1326) suggest that different perspectives throughout different disciplines cause these variations.

Even though it was suggested that internal tools are different from internal products already in the delimitations of this work, it seems that internal tools fall into the commonly used product innovation category. This is because a software tool intended for internal use cannot be described as a service, process, or technology. Instead, it fits well with the OECD's (2005) definition of *product innovation*. They define it as:

> A product innovation is the introduction of a good or service that is new or significantly improved with respect to its characteristics or intended uses. This includes significant improvements in technical specifications, components and materials, incorporated software, user friendliness or other functional characteristics.

Hence, internal auxiliary tools are product innovations. Tools developed for internal use can be divided into many other categorizations beyond the type of output. Some other categorizations that could be useful are classifications based on the environment and the purpose or aim of the innovation. However, as internal auxiliary tools can in themselves be developed and used in various environments and the aim can vary widely, deeper inspection for the multitude of categories for innovations is out of scope for this report.

2.4 Auxiliary tools in literature

This literature review was conducted using keyword searches against dozens of databases to which LUT University students have access through the ExLibris powered LUT Primo platform. All the available databases at the time of writing this thesis were used. These databases include, but are not limited to Elsevier - ScienceDirect, Springer eBooks, MDPI Open Access Journals, EBSCO - Business Source Complete, Emerald Journals, and ProQuest Central: Business/Economics. All searches were made with English keywords and English as a search filter. All searches were also limited with an availability filter to only return results that are available online.

The search began with the general search terms *internal auxiliary tool*, *internal tool*, and *internal product*. Searches were not limited to a specific part of the material. After it was realized that these terms would yield too much material that is not related to the subject of the literature review, the same search terms were used with the exact filter. This filter requires both of the words in search terms to be found from the source material. This limitation narrowed down the search results significantly. The amount of results was still in the thousands and yielded too many irrelevant results for the search terms *internal tool* and *internal product*. The term *internal auxiliary tool*, however, did not return anything.

The next round of searches was conducted with an additional filter; *software* was included as an *and* filter. This filter means that both software and the other search terms needed to be found from the material. Also, the publication date was filtered to only return results from between 1990 and 2021. This limitation halved the number of results. With the additional filter, the term *internal tools* returned about 1100 results, and *internal products* returned about 1400 results. This search showed some promise as the most relevant results returned at least one article with promising abstract, whereas none of the previous results did.

Next, the subject was limited to fields connected to internal tools or internal products. Subjects selected were science & technology, technology, product development, management, engineering, software, computer science, computer software industry, software industry, innovations, and automation. Also, all but articles and books were filtered out. After these limitations, the result numbers were lower but still high. The term *internal tool* yielded about 458 results, whereas *internal product* yielded about 725 results.

Both of these results were still too high considering that both searches had many results that referred to a phenomenon, a process, or a model instead of a software tool. Other searches filters were tried to better narrow the search but without success. The literature search platform also has a personalization feature which was turned on to narrow these searches even more. However, this led to inconsistent results when the recommendation algorithm tried to suggest sources. For this reason, the recommendation algorithm was left off.

As the search filters could not limit the source material any further, the processing was started with the citation information for the results of the two searches. All search queries are presented in Table 3. First, all result items' citation information was checked if they had either the term *internal tool* or *internal product* as a case-insensitive string. Fourteen results were found to have the term *internal tool*, and 29 items had the term *internal product*. These remaining sources' abstracts were analyzed to investigate the context in which the terms were used.

Table 3. Queries used for database searches.

Search no.	Search query	Results
s1	internal auxiliary tool (any field contains)	46,671
s2	internal tool (any field contains)	1,682,403
s3	internal product (any field contains)	2,261,722
s4	internal auxiliary tool (any field is exact)	0
s5	internal tool (any field is exact)	2,013
s6	internal product (any field is exact)	3,629
s7	s5 AND software (any field)	1,126
s8	s6 AND software (any field)	1,362
s9	s7 (with filters)	458
s10	s8 (with filters)	725

Starting with the term *internal tool*, a large number of the results were referring to something other than a software tool. Four articles referred to third-party-developed tools used internally as internal tools, and one article referred to an internal process with the term internal tool. Also, a large number of sources refer to productized internal tools when using the term internal tool. For example, Steffora (1991) uses the term *internal tool* in a context where a semiconductor company develops internally productized tools for a competitive advantage. The article indicates that these tools are developed as standalone projects and are used widely inside the company. Four other articles were found with similar references. However, none of these articles define the term in detail.

Four articles using the term *internal tool* referred to either internally used tools that are developed for specific use cases without suggestions for productization or referred to internal tools at a very general level. Crouch and Duerden (1995) used the term to describe an internally made tool that filled a specific internal use case. The article implies that this tool was developed primarily for the ongoing project without a separate development project. A second example is Bonver's (2008) article about security testing of internal tools, where they use the term *internal tool* at a very general level. This article does not separate internally made tools from each other. Another article that uses the term internal tool at a highly general level comes from an interview (Chesbrough and Euchner, 2011, p. 13) with Henry Chesbrough. They discuss sharing previously internal tools with other organizations. None of the four articles define the term *internal tool* in detail.

For the term *internal product*, there were 29 possible articles with a case-insensitive match. However, most of these sources used the term to describe whether an attribute is known only internally, describes an organization's internal process, or uses it in a completely different context. Examples of these 22 sources are an article by Ansorena, del Valle, and Salvadori (2010), which covers the internal temperature of a product, and an article by Cowan (1998), which covers the internal globalization processes of a product. In turn, Nambisan S. and Nambisan P. (2008), Sherwood and Covin (2008), and Teresko (2004) used the term *internal product* to refer to traditional closed innovation product development.

Some sources seem to use the term in the context of software tools, but further examination of the text reveals them not to have the same context. In their article, Wallin (2012) investigates a case study where internal knowledge leads a company to extend their service offering from maintenance to other external service offerings. Wallin uses the term *internal product knowledge* without defining it further. The rest of the article confirms that this wording is used only to describe an organization's internal knowledge about products in general. Likewise, based on the abstract alone, Zhang, Tang, and Wu (2021) seem to investigate the internal product scope as in the internal software tool scope. However, later in the article, the term *internal product scope* is used to reference the extent of a firm's product portfolio within the industry.

The rest of the potential sources use the term *internal product* in a context that indicates the tools are developed as standalone projects and are used widely inside the company. One example of these is an article by Ahonen and Savolainen (2010, pp. 2176–2181), which explores a case of an internal tool development project. In this article, the development project is separate from other projects, and the output was intended for internal usage. None of these seven projects define the term *internal product* in detail.

2.5 Internal tools and internal products

Section 1.3 already provided a fair preview of this section by presenting the scope of this study. The concept of internal tools mainly determines this scope. However, not all of the given limitations are necessarily properties of internal tools in general. For the clarity of the study, only software-related tools are explored and defined in this section.

As discovered in section 2.4, the terms *internal tool* or *internal product* do not have a well-established definition currently. In most cases, the union of these words does not refer to a tool developed for internal use. However, multiple different names and meanings can be found across different mediums for the same basic concept. When the terms refer to internally used software, the common denominator for these cases is that internal tools and products are used only internally. Beyond this common denominator and the context of software, these do not seem to have much in common.

Literature did, however, reveal common traits for internal tools and internal products separately. More specific uses of the term *internal tool* suggested that internal tools are developed primarily for specific use cases. These articles also indicate that internal tools are not developed as a standalone project but in other projects as helpers. Articles with the mention of *an internal product*, however, indicate a standalone project. Internal products also seem to be developed for broader use from the start than internal tools. Hence, these tools are productized from the beginning.

Despite the small number of mentions in academic literature, internal tools and products are not without a presence in real-world. Many recent columns, blogs, and articles have brought internal tools to people's attention. Retool, a company specializing in internal tool building solutions presented its findings for a public survey conducted in early 2021 (Garcia, 2021). The context in which Retool uses internal tools in their marketing material and their summary of the public survey matches on a general level the articles that used the term for software tools. The same applies to another internal tool specialized company, Budibase's, marketing article about internal tools (Johnston, 2021) and a company called Internal's mission statement on their homepage (Internal, 2021). These companies do, however, count externally developed tools as internal tools or products in some cases.

The authors used these terms vaguely or did not define them separately in all found literature and other sources. Excluding the mentions where authors used the term *internal tool* or *internal product* in a completely different context, the remaining mentions can be divided into three categories. Third-party developed tools, internally developed tools for specific projects as a part of the same project, and internally developed tools for general internal use as a standalone project. In all cases, like the general definition, the scope of these tools or products is missing or is vague.

These categories, in turn, are built from different types of auxiliary tools or products. The third-party-developed tools are either unchanged third-party products that have been configured to suit the organization's use cases or are not customized at all. These include products like different dashboards, spreadsheets, use-case-specific solutions, and generic software solutions. This study will use the term *third-party product* for this tool type. Another significant and easily recognizable category is products developed inside an organization for general internal use as a standalone project. These include third-party products that have been modified beyond simple configuration internally. Examples of these tools are internal portals and internal applications widely used throughout the organization. This study will use the term *internal product* for this specific tool type.

In turn, defining internal tools is much more challenging. The literature indicates that internal tools are more specific than internal products and are not developed as a standalone project. A problem arises when a project consists almost entirely of a tool developed for the same project but a general use case. Ultimately, the intent defines which tools are internal products. The intent to productize an internal tool makes it an internal product. The easiest way to outline the definition of internal tools is to check if the internally used auxiliary tool is an internal product or a third-party tool. If the tool is neither, it is an internal tool. The relationships between these tool types are illustrated in Figure 4.

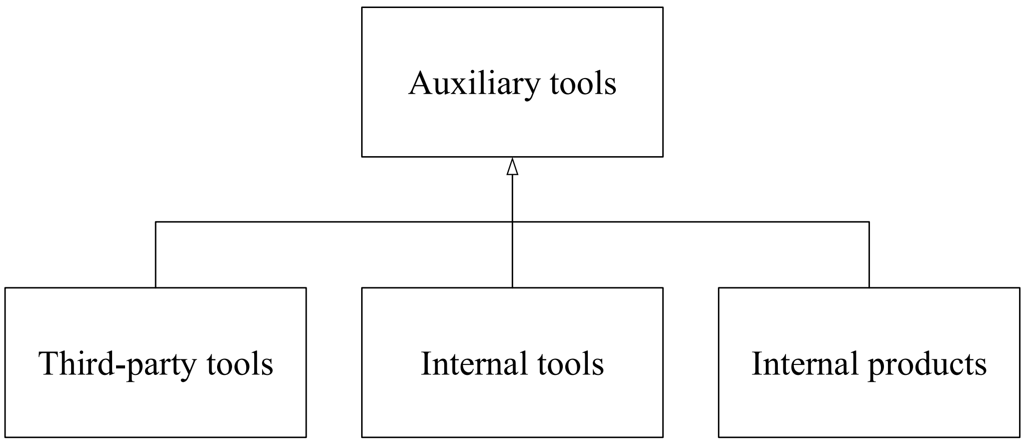

Figure 4. Internal auxiliary tool types.

Even though the literature did not provide one comprehensive definition for internal tools or products, different auxiliary tool types can be differentiated. From these bases, a definition for *internal tools* is proposed as such: internal tools are auxiliary tools developed by employees inside an organization for specific internal use cases without productization purposes, internal or external. Unmodified third-party tools and open-source projects, simple derivatives of third-party tools or open-source projects, and software languages, libraries, frameworks, or protocols are not internal tools.

From the new definition and its bases, some deductions can be made. Internal products are usually developed as their standalone project and internal tools as part of another project; internal products' budgets are also more precise than internal tools' budgets. Internal tools may also not be the primary focus of the project. As internal tools are developed, at least at first, for a single project, they also might not be as polished as the result of a standalone development project.

Still, the difficulty of separating internal tools from internal products should not be disregarded. As the primary separator between these tool types was the purpose, the situation can change over time. After all, the tools can be developed beyond the original project for which it was developed. If the purpose changes from a specific helper for an ongoing project to productized tool for internal use, the internal tool can evolve into an internal product.

3 INTERNAL TOOL DEVELOPMENT IN ACTION

The previous chapter defined internal tools on a theoretical level. Real-world examples need to be examined further to study the process of internal tool development and its effects. This chapter sheds light on internal tools with the help of two qualitative case studies.

The case studies presented in this chapter are from the same SaaS company specializing in Enterprise Content Management or ECM, later referred to as the Case Company or CC. The CC has its own ECM product for which the CC, along with its partners, offers deployment and maintenance services. This solution will be referred to later as the Case Company Solution or CCS. Employees of the CC have developed a wide range of different internal tools over the past decade. Internal tools have been developed in various departments and for multiple purposes, ranging from small personal scripts to tools that have grown to internal products and used company-wide.

Both of the cases focus on internal tools developed for one more extensive software delivery project that was active for three years. Due to the challenging requirements and special nature of the project, it was known from the start that custom processes or tools could be needed. For this reason, the project team included many people with a high-level of technical knowledge. Even though both tools were developed for the same project, the developers of these tools were different people with completely different areas of responsibility. However, as internal tools usually are, these tools were developed as part of the delivery project, not as standalone projects. For this reason, there were no detailed time bookings for the development work specifically, only estimations. All estimations presented later in this chapter are based on the best estimations of the responsible project manager and other team members. It should be noted that these estimations only visualize the scale of the development processes and are not used as detailed empirical evidence as is.

3.1 Empirical research theory

As mentioned in section 1.2, this work uses a case study methodology to explore the real-world examples of internal tools. Defining and using the term *case study* can be difficult because it is used across different disciplines with varying meanings (Mills, Durepos, and Wiebe, 2010, p. xxxii). A general definition for a case study could be summarized to be an in-depth examination of a specific real-world phenomenon with clear conceptual and empirical boundaries and substance that is leveraging qualitative research methods (Crowe *et al.*, 2011; Feagin, Orum, and Sjoberg, 1991, p. 2; Orum, 2015, p. 1509). Yin (2003, p. 1) proposes that case studies should be favored when *why* or *how* questions are posed, when the focus is on phenomena with real-world context, and when the researcher has limited control over that phenomenon. Yin (2003, p. 13) also proposes that the boundaries between the case context and the phenomenon usually are not clear or evident.

Different authors have categorized case studies differently as there are multiple definitions for the term *case study*. Yin (2003, pp. 1) has divided case studies into three separate categories: descriptive, exploratory, and explanatory case studies. *Explanatory case studies* explain some conditions related to the case study, while *exploratory case studies* identify new studies' ideas (Yin 2003, p. 1–17.) The third category, *descriptive case studies*, describes a particular phenomenon (Yin 2003, pp. 1–17). Another commonly referenced classification is proposed by Stake (1995, pp. 13–138). They split case studies into intrinsic, instrumental, and collective case studies. They call *intrinsic case studies* that are executed to understand a particular case better. On the other hand, *instrumental case studies* are undertaken mainly to gain insight into some other issue or phenomena. *Collective case studies* are composed of multiple cases that resemble instrumental case studies.

To explore the value of internal tools to software delivery projects, they need to be examined in the context of a real-world delivery project. Like Yin (2003, p. 13) proposes, these cases do not have clear boundaries between the phenomena and their context. The main research question defined for this thesis asks a *how* question: how can the value of internal tools be measured. For these reasons, case study methodology will be utilized for uncovering answers to the research questions. As two different cases are used to describe internal tool development and gain insight into the value of internal tools for software delivery projects, these cases can be categorized as descriptive case studies or as a one collective case study.

Other research strategies or methods were considered for this thesis while collecting preliminary information. However, the fact is that internal tools are usually developed by a small team or a single person for the problem at hand. This fact makes it a phenomenon that is hard to observe in real time. Quantitative methods were not feasible as it is challenging to find data for a quantitative approach when the separation of internal tools and products is not widely recognized.

3.2 Case: Deployment Tool

Some background knowledge for the case company and the case company solution itself needs to be reviewed to understand the use cases for the Deployment Tool. The CC has targeted its solution towards small to medium-sized organizations from the start. Despite this target market preference, the CCS is used in many larger organizations as well. This expansion to different customer bases has brought some disadvantages alongside the apparent advantages.

The CCS consists of various applications with different functions. The server application handles the centralized processing and storage for the solution. The most visible part, the client application, interacts with the end-user and communicates with the CCS server. For administrators, there is a separate application that manages the structure and the functionality of the CCS itself. The CCS as a solution is a metadata-driven, system and repository neutral, and customizable ECM system. Everything from documents to information is stored in a centralized data repository called a vault. The CCS server can host multiple vaults simultaneously, but despite the vaults joint hosting, every vault is entirely independent of other vaults. This separation has significant benefits in security and categorization of information. As a disadvantage, configuring and maintaining multiple separate independent vaults requires more effort when compared to a more straightforward, less divided approach. In addition to increased workload, repeated manual configuration steps increase the risk of misconfigurations or mistakes. These disadvantages multiply with larger customers that usually also have more vaults.

One critical use case for maintenance or deployment projects has been moving vault structures, content, and configurations between vaults. Especially with larger projects where vault count grew to dozens of vaults, the deployment process from a development environment to quality assurance and from quality assurance to production reached a point that made manual deployment impractical. This impracticality does not come only from the time perspective but also from the quality assurance point of view. Manual deployment takes a lot of time, and since the deployment process has multiple steps, potential for errors increase with every added vault and added deployment step. The Deployment Tool was created to eliminate or at least relieve these problems. The delivery project behind the cases specifically had such a tight schedule with so many vaults that it was impossible to deploy vaults between environments manually. This need pushed the development of this tool into motion.

The need for the Deployment Tool was realized during the initial delivery project. For this reason, it was developed alongside the project. This dynamic development placed most of the development focus on creating a working Minimum Viable Product or MVP as fast as possible. The effort put towards creating an MVP can vary significantly depending on the case (Owens and Fernandez, 2014, pp. 96–97). In this case, the MVP had to demonstrate the functionality of each core use case to reassure the project. The development needed to move quickly for the tool to have this level of MVP ready on short notice. This led to the situation where the scope of the tool was not clearly defined when the development started but instead derived purely from the project's apparent needs. These were also the minimum requirements for the MVP.

According to Pohl and Rupp (2015, p. 8), requirements for software development can be identified as belonging to three separate groups: functional requirements, quality requirements, and constraints. Functional requirements define the functionality that the software offers. Quality requirements, on the other hand, focus on specifying the desired qualities of the system. Quality requirements determine how functional requirements should be fulfilled and usually affect the system's overall architecture more than functional requirements. Unlike functional and quality requirements, constraints do not need to be implemented; they need to be adhered to during the development. The constraints provide limits in which the other requirements are implemented. (Pohl *et al.*, 2015, p. 8.)

From the standpoint of the functional requirements, only the most urgently needed use cases were included in the MVP. These were authentication to the vault, replicating content and structure between vaults, installing vault-specific custom applications, and copying configurations from one vault to another. Figure 5 demonstrates the use cases for the initial version of the tool.

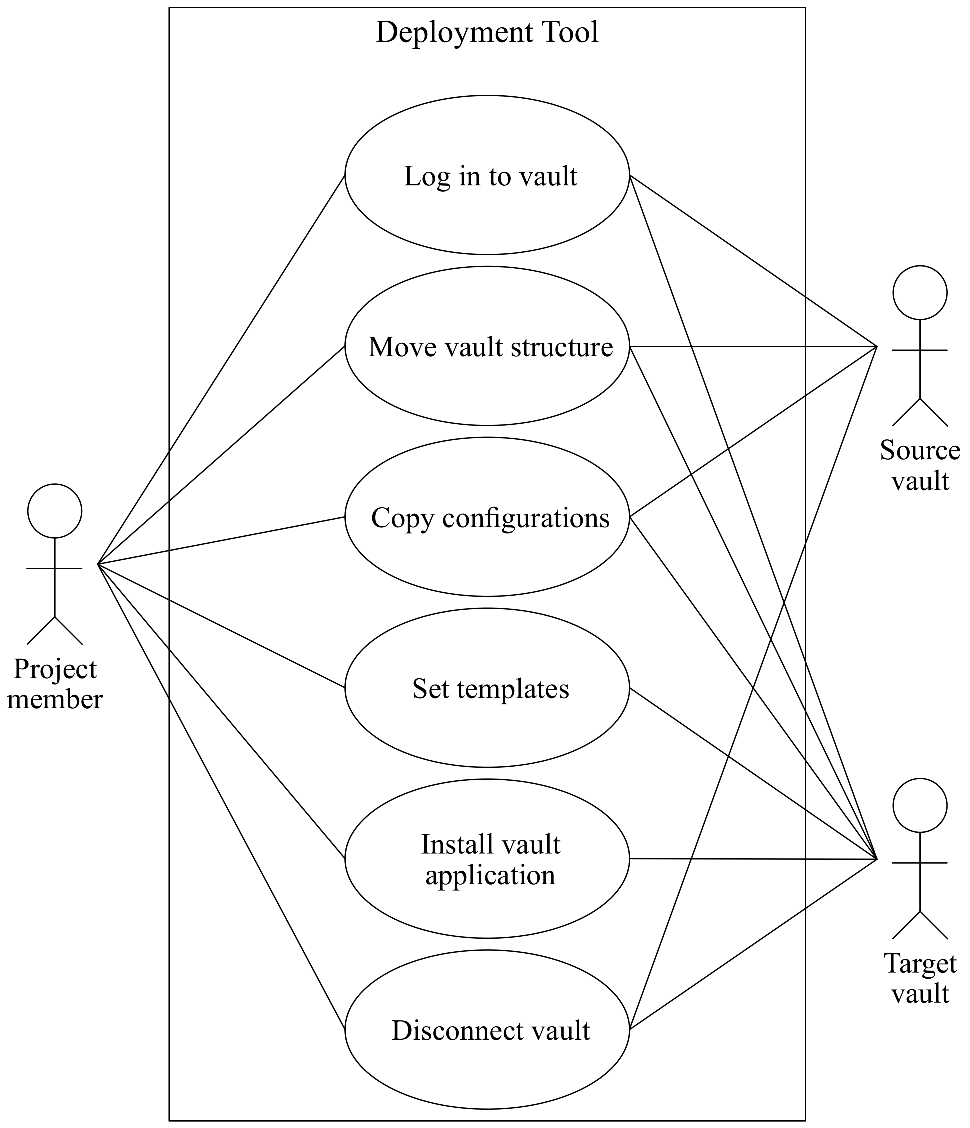

Figure 5. Deployment Tool use cases.

Quality requirements were also derived from the delivery project's demands. These requirements could be categorized into three groups, performance-related requirements, usability-related requirements, and requirements regarding transparency. Performance requirements arise from the tight schedule for these releases. Requirements for concurrent release to multiple vaults and automated action execution were critical from a time savings perspective. On the other hand, usability requirements came from the fact that deploying a release required multiple steps. Multiple different steps increased the number of needed settings for the tool, increasing the complexity. To combat this problem, deployment actions performed by the tool needed to be modular and easily repeatable. In addition to various options, the fact that Deployment Tool had to be run by numerous people during the project added more requirements. These include the need for standalone executable format and saveable, transferrable, and restorable configurations.

The last category for quality requirements is transparency. When performing actions to a production vault, the confidence in the tool must be high. This fact leads to requirements for progress tracking and logging. As complex deployments consist of many items that need to be tracked, the tool needed a simple user interface or UI to fulfill these transparency requirements. Table 4 summarizes the quality requirements for the Deployment Tool.

Table 4. Quality requirements for the Deployment Tool.

Requirement category	Quality requirement
Performance	- Release to multiple vaults at the same time - Multiple actions can be performed in a row without user interaction
Usability	- Standalone application - Actions are modular - Actions are easily repeatable - Configurations can be saved and restored
Transparency	- Detailed logging - Release progress can be tracked from the UI

The development of the Deployment Tool was initially started by one project member who saw the potential in automated deployments. This project member was also frustrated at the slow and tedious manual processes involved in the manual deployments with the growing number of vaults. This project member had an excellent understanding of the CCS and had practical development skills such as coding experience and knowledge about network technologies. Furthermore, this employee had also experience in developing internal tools at the CC. Additionally, the internal culture at the CC also played a part in the initiation of the development. The internal culture did not hinder developing a new tool as creative problem solving and developing internal tools were encouraged at the CC.

In addition to the project member's frustrations, it had also become a problem that the manual releases did not go as planned. Due to multiple complicated manual steps being performed on multiple vaults, the manual deployments had regressions and mistakes. Without action, the project would most likely not be finished with the original budget and schedule. An idea for a solution, the time pressure faced, the inadequacy of manual deployment, qualified team members, and the tendency for innovation at the CC led to the idea's implementation.

The initial delivery project used the Deployment Tool heavily for the rest of the project. According to the delivery project manager, the project used this tool for over 1500 vault-to-vault deployments. The deployment model consisted of three environments, development, quality assurance, and production environment. Each environment has 35 to 40 vaults. At the start of the project, full deployment of a single vault from one environment to the next took an estimated 30 to 45 minutes. With the introduction of the Deployment Tool, this single vault deployment time was shortened to less than 10 minutes. As the delivery project had over 20 new full releases through the whole deployment process and about a dozen development-related vault-to-vault releases, the Deployment Tool was estimated to have saved at least 760 hours of work in the delivery project alone. Table 5 shows the rough estimation for work effort at a minimum for the project.

Table 5. Estimated work effort saved by the Deployment Tool in the delivery project.

Description	Estimated time required manually (h)	Estimated time required with the Deployment Tool (h)	Estimated time saved with the Deployment Tool (h)
Full release for one vault	0.5	0.1	0.4
Full releases	760	152	608
Separate development releases	190	38	152
Total	950	190	760

As the project team that needed this tool also developed it, its effects need to be explored beyond pure time savings. One of these effects is the added workload for the project team. According to the project manager of the delivery project and project notes, actual development took at least two weeks for the minimum viable product. As the project members started the development with minimum design, the tool needed some refinement and maintenance during the delivery project. This additional effort is estimated to be around another two weeks or 75 hours. When this development time is subtracted from the total estimated saved time calculated in Table 5, it is clear that in this case, the tool saved a considerable amount of time even when the development time is taken into consideration.

In addition to clearly time-related reasons, the Deployment Tool was created for quality assurance. As mentioned before, regularly repeated complicated manual steps can lead to errors. These errors can, in the worst case scenario, erode the confidence in the project and sometimes lead to situations that are difficult to fix. Even if the Deployment Tool can also be misconfigured, when the steps are standardized and the same configurations can be used repeatedly, or as a template, the chances for errors are significantly lowered. As the errors in the manual deployments can be time-consuming to fix, any errors mitigated will also lead to time savings. However, these time savings can be hard to track as the number of mitigated mistakes cannot be known. For this reason, these possible time savings cannot be estimated for this tool.

However, as previously mentioned, the project was in danger of failing to meet the initially planned budget and schedule. In the worst case, this could have resulted in an abrupt end for the project. Even when there could have been other ways to make the deployment process more robust and efficient, it is apparent that the Deployment Tool also enabled the project to move forward.

After the delivery project, the same customer environment followed a regular, monthly deployment cycle for new features and maintenance-related changes. It is likely that these maintenance deployments could not be executed in their current form if the Deployment Tool was not in use. From that point of view, the enablement the tool has provided outweighs all benefits from time saved. Still, comparing the savings from a more extended period shows what a significant impact this tool has had on the particular customer case. These regular maintenance deployments have been ongoing for over three years, bringing the number of full releases up to 60. A quarter of these maintenance releases included development-related vault-to-vault releases. These maintenance releases increase the estimated saved work time hours to over 1900, which amounts to over 250 saved workdays. Table 6 shows the estimated total work effort saved by the Deployment Tool for this particular customer case.

Table 6. Estimated total work effort saved by the Deployment Tool for one customer.

Description	Estimated time required manually (h)	Estimated time required with the Deployment Tool (h)	Estimated time saved with the Deployment Tool (h)
Full release for one vault	0.5	0.1	0.4
Project related deployments	950	190	760
Development time	0	75	-75
Maintenance releases after the delivery project	1520	304	1216
Total	2470	569	1901

After the initial delivery project, the Deployment Tool has been adapted to multiple other projects with additional work done at the CC. It is still developed solely for internal use, but this expanded use has required additional development to enhance the usability and make it more reliable for different projects. The tool has gained support for many new use cases and also received a significant makeover to the UI of the app. As various persons nowadays develop the tool in the organization and resources are allocated for its maintenance, the Deployment Tool has become an internal product at the CC. New projects are planned from the start to rely on this tool if even the current, enhanced built-in options are not enough, proving its value in the organization.

It was noted in the last paragraph that the value for developing and using an internal tool is, in the case of the Deployment Tool, clear. Still, the perceived value should be explored more closely to better understand the tool's degree of value. In the case of the Deployment Tool, everything that the tool did could have been done manually in theory. Automating repetitive tasks, however, saved a substantial amount of work hours. The modular nature of the tool also made it much more repeatable. As previously touched on, the delivery project had a tight schedule for the whole project and required that releases needed to be done outside of office hours. The growing number of vaults and release-related tasks made it impractical to only move forward with manual releasing methods. Also, the faster the project would be completed, the sooner the customer could also start using the CCS.

Another trait of the Deployment Tool for the delivery project was that it increased the release quality by decreasing mistakes. These mistakes, in theory, could have been eliminated with more manual testing and checks. For this reason, the increased quality can be considered as saved work hours for the project. As the project needed and valued time savings highly, the fact that the Deployment Tool saved release-related work hours at least by 80% and decreased the need for additional tests and checks, the tool provided a high level of value to the project.

Even more critical than finding an efficient way to complete the project is to complete the project in the first place. For this delivery project, it meant keeping a tight rein on the schedule and budget. For this reason, the most valuable attribute of the Deployment Tool for the initial delivery project was that it enabled the project to be completed. Even when there probably would have been some other solution that would have made this delivery project possible if this particular tool would not have been first thought of and then implemented, it does not lower the importance of the Deployment Tool in this case.

3.3 Case: Migration Tool

The context for this case study is similar to the case of the Deployment Tool. Even though it was developed for the same delivery project as the Deployment Tool, the Migration Tool was developed for a different part of the project. The project was also set to migrate documents and other data from an old system to the CCS. The number of documents was considerable, up to 5.5 million documents with over 15 million versions, totaling over 40 terabytes across 15 different vaults. Some of the data was still used in the source system and needed to be migrated multiple times, even after the initial delivery project ended. Processing a large amount of data always brings some challenges. However, the true challenge was that the documents had metadata or information describing this document in the old system.

Gartner (2021b) defines *metadata* as information that defines several aspects of an information asset in order to increase its usability over time. In this case, the metadata had information about who was the document's owner, when it was created, and hundreds of other pieces of information related to the document. This metadata can exist in different systems as different parts of the system structure. Even when the CCS is a metadata-driven ECM, migrating a large amount of metadata from one system to another is a complex project.

In the CCS, all information is tied to objects. These objects have different customizable types. Every object type represents a different concept in the vault. For example, a document is a built-in object type with certain built-in defaults designed to store documents. These objects have customizable classes that represent different types of previously mentioned objects. One example could be a report class that is one type of document. Each of these classes can have customizable properties that make up most of the object's metadata. Objects do have some built-in properties, like creation and modification time. Objects can have other objects as part of their metadata, forming a relationship between these objects. Later in this chapter, these objects included as metadata for other objects will be referred to as *supporting objects*.

Contrary to the Deployment Tool, the decision to develop a new internal tool for this case came up reasonably quickly. The sheer size and the timetable for the migration were too demanding for fully manual importing from the start. For scale, importing documents with an already existing, general use internal product took about one hour for 2000 documents with no previous versions. Every version would take about as much time as a new document. Migrating these 40 terabytes would take, even in the best case, over 96 full days. There were also other limitations with this pre-existing internal tool, but the lack of speed already excluded it from beign a viable option. As with the Deployment Tool, there were no third-party-developed options on the market or exact built-in options either.

However, many other built-in CCS features, third-party-developed tools, and internal tools or products could help with the migration. Many of these were in regular use for other migration projects at the CC. Still, none of these could control the whole complex migration process. A manual process with these helpers was not an option either, as the process needed to be repeated for multiple vaults, for some of them multiple times. Manual control of the migration process would increase the risk for errors in the migration. It would also be much slower than a fully or partially automated process. Other data handling and processing requirements for the migration presented by the customer made it clear that a new tool would be needed for this project.

The functional and quality requirements were considerably more specific and also more extensive than with the Deployment Tool. The migration needed a much more customer-specific solution. Also, as the tool's development was started, some more minor use cases were included in the tool's scope for convenience. This case study focuses on the primary function of this multifunctioning tool, data migration. The source system metadata needed to be transferred to a separate database—this transferred metadata needed to be validated and prepared for the migration before actual migration could start. Validation is needed to ensure that all needed information can be found from the migration database and that the data is intact. Another requirement was that the data needed to be prepared for the migration by mapping it to correct properties in the CCS. Some of these mappings required supporting objects in the vault and these supporting objects needed to be created in the vault before document migration could start. After preparations, documents should be migrated to the vault. To ensure all data is migrated successfully, the migration needed to be validated. These were the primary functions of the tool: validating the data used for migration, preparing data for migration, importing support objects to the vault, importing documents to the vault, and validating the migration. Figure 6 demonstrates the primary functional use cases of the Migration Tool.

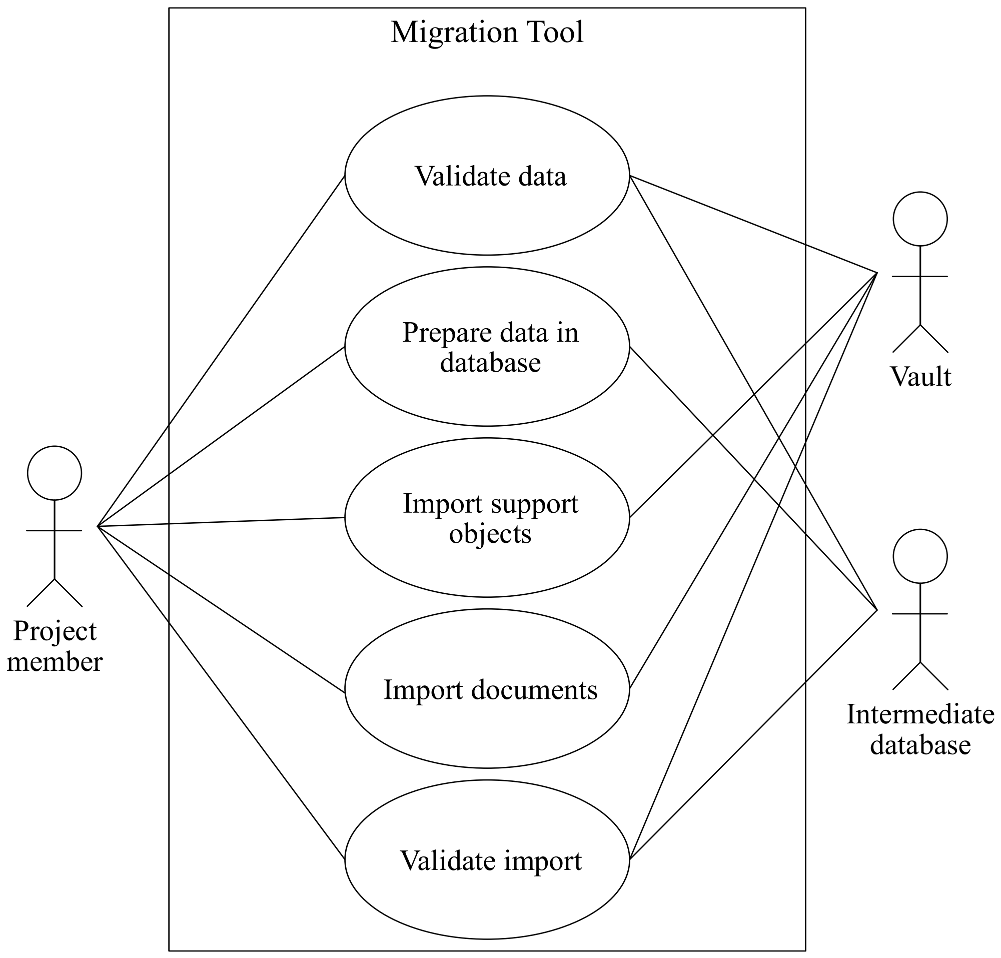

Figure 6. Migration Tool use cases.

As with the Deployment Tool, quality requirements for Migration Tool can be categorized into three groups, performance, usability, and transparency-related requirements. As the amount of metadata was huge, the project decided that most of its changes should be done in the database. Databases are optimized for batch operations; moving the processing to a separate application would be unnecessary. From a scheduling perspective, some vault migrations needed to be done during the same weekends. For this reason, the tool needed to be able to migrate multiple vaults at the same time. These time constraints dictated that the tool should be able to perform multiple steps without user interaction. If the tight-scheduled migration stopped for a night because nobody was at the computer, valuable time would be wasted. These requirements belong to the performance-related quality requirements.

The usability-related quality requirements were mainly the same as with the Deployment Tool but for slightly different reasons. The Migration Tool needed to be a standalone application to control all of the steps. Actions of the tool needed to be modular and easily repeatable to run them in different configurations in different vault migrations. These requirements for repeatability and modularity also required the configurations to be saveable and restorable.

For the same reason as with the Deployment Tool, transparency of the tool's actions was essential. However, with the Migration Tool, the need for transparency was highlighted as this tool controlled other tools. Furthermore, it also performed many more minor actions that it would be hard to spot or troubleshoot any possible problems without these requirements. These requirements included the need for detailed logging and progress tracking from the applications UI. Also, migrations would need to be verifiable from the vault itself. Even when the database would have the source material, and other tools were not reporting any problems, this still did not mean that everything had worked as planned. The migration could be considered completed only when the results were verified from the target vault. Lastly, any changes or mappings built on the source data should be separated from the original data. This separation ensures that fixes can be performed even if something goes wrong in the migration. Table 7 summarizes the quality requirements for the Migration Tool.

Table 7. Quality requirements for the Migration Tool.

Requirement category	Quality requirement
Performance	- Most of the changes in metadata is processed in the database - Multiple environments can be migrated at the same time - Multiple actions can be performed sequentially without user interaction
Usability	- Standalone application - Actions are modular - Actions are easily repeatable - Configurations can be saved and restored
Transparency	- Detailed logging - Release progress can be tracked from the UI - Migration needs to be verified from the vault - Changed or processed data is separated from source data

Although the project was unanimous about the need for new tools, one project member took the responsibility of developing this tool. This person was an expert at data analytics and with an excellent understanding of the CCS. This project member also had practical development skills such as experience in coding and data modeling. Even when the Migration Tool was probably not their first internal tool developed at the CC, it was one of the most extensive internal tool projects at the CC at that time. For this reason, other employees at the CC were involved in the development also. Still, many of the technical choices made during the design and development of this tool reflect the lead developer's expertise and interests. As there were no extensive best practices established for such customized data migrations, the custom tool developed for this purpose had the freedom to choose technologies and methods best suited for this particular case.

The Migration Tool, in conjunction with the Deployment Tool proved to be the backbone of the delivery project. Even when the project had many other excellent tools developed specifically, this highly customized migration would not have succeeded without the Migration Tool. According to the project manager, the development took a considerable amount of time from the total work hours of the project. As the tool was vital for the project, it is hard to separate work hours for the tool development from other migration-related tasks. They estimated that the MVP took 1–2 months of development work, totaling over 450 hours as a rough but cautious estimate. Even though this is a considerable effort, it seems much more reasonable when compared to the alternatives.

As previously stated, fully manual migration would not have been reasonable due to unreasonable work effort and complexity. The migration needed to run multiple data processing steps for validation, mappings, and preparations. Doing these by hand would have taken multiple hours of work when the Deployment Tool automated most of these tasks to a few clicks. The actual data migration would have been a very manually intensive process. The project members would have needed to operate multiple, more specific tools; process large datasets; move files from one server to another; and perform checks during and after the migration to ensure everything worked as usual. The Migration Tool handled all of this, leaving only smaller checks and configurations to the operators. Even if exact estimations are impossible in this case due to the complexity of the process, employees at the CC who were using the tool after the initial migrations gave a conservative estimation. They estimated that all of these steps would take at least 28 hours of work if done manually, per one vault migration. With the Migration Tool, this took about 7 hours per vault. These estimations will use this general time estimation per vault as the baseline for all imports. It should be noted that these estimations do not contain all steps for the migrations. Estimations include only hours directly related to the migration process executed either by manual actions or with other existing tools.

The initial migration consisted of 15 migration vaults that needed to be migrated. For each vault, there were additional 3–4 delta migrations on average. Some of these were much more resource-intensive than others due to their complexity and size. In addition to these, some iterations needed to be executed at the beginning of the project. Also, a subset of the data had to be imported to some quality assurance vaults. After the initial delivery project, there were six additional delta migrations to the production environment.

Considering all these steps, it would have taken 2772 hours or about 370 workdays to finish the migration project manually. The migration Tool took an estimated 1143 hours to finish the migration project. Total saved time just considering this initial delivery project would then be 1629 hours or about 220 workdays. Estimations for work hours of the initial delivery project migrations can be found in Table 8.

Table 8. Estimated work effort saved by the Migration Tool in the delivery project.

Description	Estimated time required manually (h)	Estimated time required with the Migration Tool (h)	Estimated time saved with the Migration Tool (h)
Full migration to one vault	28	7	21
Initial migration to production vaults	420	105	315
Delta migrations to production vaults	1680	420	1260
Migrations to quality assurance vaults	224	56	168
Development migrations	420	105	315
Migration Tool development	0	450	-450
Total	2772	1143	1629

The Migration Tool needed much more configuring than the Deployment Tool after the working MVP was in use. The prototype was completed reasonably quickly but making other tools work seamlessly with the Migration Tool needed further testing and configuring. Both the Migration Tool and the Deployment Tool controlled their respective processes, but the Deployment Tool needed to interact only with the CCS. The Migration Tool used multiple scripts, standalone internal tools, and third-party tools to achieve the desired result. The development time for all separate standalone tools or scripts is not included in the work time estimation as some already existed, and some could have been used and developed for the manual process.

As briefly touched upon in previous paragraphs, developing the Migration Tool was in itself a significant undertaking, totaling up to 450 work hours. The tool was developed by the same project that used it, as with the Deployment Tool. For these reasons, the development time is a crucial variable when considering the effects of internal tools on the project. If the time used for development is greater than the time saving achieved with the tool, the purpose for its development needs to come somewhere else. In this case, however, the delivery project estimations already show that the time saved during the project makes it a valuable addition to the project for this reason alone.

In addition to time savings, this case is another excellent example of situations where an internal tool can enable projects or other undertakings. Even if many repetitive and complex things can be done manually, increasing labor costs and slow progress can cancel the project. In the case of the Migration Tool, the initial migration project needed a custom tool and would not have been viable without it. The initial delivery project was already a large project for seemingly simple tasks, and brute-forcing the migration by manually repeating complex activities would have ballooned the project's budged excessively. The fact that the migrations were completed on schedule makes the Migration Tool already valuable.

As was with the Deployment Tool, one of the essential functions of the Migration Tool was quality assurance. Automating tasks and checks ensured a higher quality result for the migrations by eliminating mistakes due to manual error. Fewer mistakes also translates to time savings for the project. These savings are impossible to estimate in a single case study. For this reason, the quality assurance viewpoint will not be estimated to the time savings.

During the additional delta migrations, the tool was not developed further but used as-is. Even though the tool's core use cases were well documented, the fact that it was developed for a particular purpose and included complex logic did reduce the efficiency of the other delta migrations after the operator from the delivery project left the CC. This inefficiency was amplified because the original operator was the principal developer for the Migration Tool and usually operated the tool themselves. Still, the additional need for the tool made it even more valuable than it was before and raised the estimated total hours saved to 1965 work hours. Table 9 shows the estimated total work effort saved by the Migration Tool for this particular customer case.

Table 9. Estimated work effort saved by the Migration Tool for the customer.

Description	Estimated time required manually (h)	Estimated time required with the Migration Tool (h)	Estimated time saved with the Migration Tool (h)
Full migration to one vault	28	7	21
All migrations during delivery project	2772	693	2079
Additional migrations	420	105	315
Migration Tool development	0	450	-450
Total	3220	1255	1965

As the delta migrations for this specific customer were completed, the demand for the Migration Tool was significantly diminished. The main reason for this was that the tool was developed for a particular purpose, and adapting it to another large migration project would require significant changes to the source code. Also, the environment in which the migration was performed was highly customized. Building or replicating a similar environment would take significant effort. It could be argued that the precise nature of the tool and the environment would mean that any adaptation would be a new separate tool. Secondly, the best practices for large migrations have evolved since the Migration Tool was initially developed substantially due to this delivery project. All of these reasons combined means that the Migration Tool will probably be used only for this specific customer environment.

Despite the specific nature and being tied to a single customer environment, the Migration Tool provided immense value for the delivery project. Not only did it save an estimated 1629 hours of work compared to the manual process, but it also enabled the whole project in the first place. As the project was on a tight schedule, time savings were the tool's most important benefits after enabling the project itself. The achieved time savings were enormous, even considering the significant development time, due to the repetitive tasks that could be automated. Furthermore, although the quality assurance could not be estimated as time saved in this case study, considering the complexity of each vault migration and delta migration, the tool decreased mistakes in the migration process. The tool was also a crucial part of the project. Overall, the tool provided a high level of value to the project.

4 INVESTING IN INTERNAL TOOLS

Cambridge Dictionary (2021) defines *investment* as using resources such as money, effort, or time to make a profit or get an advantage. By this definition, internal tools can be considered as investments. Chapter 3 explored two typical cases of internal tools in depth. With the help of previously mentioned cases, this chapter explores the prerequisites for internal tool development and what type of effort the development takes.

4.1 Spark for internal tool development

As previously noted in section 2.1, Brown *et al.* (2011, p. 381) described the innovation process as starting from the space of inspiration where the motivation for searching for a solution arises from a problem or opportunity. The development of the Deployment Tool started from frustration about the cumbersome nature of manual vault deployment and the problems which arose from manual deployments. The number of tasks needed to be run in exact order was considerable, and when the number of vaults to be deployed also increased, this problem also grew. Another problem was the time pressure for the deployments. The deployments needed to be done on a tight schedule, and later in the delivery project, it became apparent that the manual process would not be fast enough.

Luckily, there were also opportunities that made custom tooling possible for the use cases of the project. All of the steps which were cumbersome were also automatable. The CCS API, which enabled this opportunity, allowed the remote control of the tasks that needed to be run on the CCS server. Without this possibility, no tool could have orchestrated the tasks that allowed this process to be automated. Automation, in turn, increases the efficiency of the migration, easing the scheduling problems and reducing manual tasks. Another important opportunity was the repeatable nature of the deployment process; the same tasks needed to be run against the same vaults many times during the project. If the process was expanded to other vaults, the tasks were still the same. This repeatability enabled the automated process to be efficiently run time after time without significant changes to the process.

On the other hand, the Migration Tool was started in the early stages of the delivery project, as manual migration would not be viable. It was realized early on that the scheduled time and work hours would not be enough to complete all vault migrations with specified custom requirements without a tool that controlled the process. In addition to the time-related problems, manual migration would not be reliable enough to complete the project successfully. Thankfully, the case of the Migration Tool also had opportunities that enabled the tool development. Existing tools needed for the migration process were able to be operated by the Migration Tool. Also, the data was in the database, which was also accessible by the Migration Tool. This access enabled the Migration Tool to automate the migration process. Also, as with the Deployment Tool, the use cases for Migration Tool needed to be repeated multiple times and could be controlled by saveable configuration files. This repeatability enabled the Migration Tool to increase the efficiency of the migration process considerably, reducing the needed time for migrations. Repeatable tasks would also efficiently increase the quality of migrations by obediently repeating steps that have already been proved to work.

There were apparent problems for the current situation and opportunities for the new tool to solve these problems in both cases. For these cases, the time savings provided by the tools were the most important reason for the start of their development. Repeatability in both cases played an important part also. However, this attribute can also be associated with time savings as the repeatability ultimately made it possible for the tools to save a considerable amount of time. Even though the project team created both tools, the time estimations afterward proved that the development ultimately saved a large amount of time. This benefit would not have been the case if the development had taken considerably more time. For this reason, the effort needed for the development also plays a significant role when considering the development of a new internal tool.

Even though the potential time savings were the most important reason to start the development of both tools, it is clear that they did not only save work hours or help the schedule, they enabled the delivery project to be completed. Without the Deployment Tool, the problems encountered during the start of the project would have continued and may have caused significant problems for the project or even an abrupt end. According to the delivery project manager, the Deployment Tool was an essential part of the project, so much so that they think that the tool would have been developed even if it had not provided any time savings. They also estimated that there would have been an evident spark for the development of the Migration Tool even if the use of the existing tools would have been more time-efficient. The sheer amount of files and metadata was so huge, and the requirements from the customer so specific, that making the migration successful would not have been possible without the Migration Tool.

Based on these findings, it is evident in these cases that the spark for their development started from the motivation for searching for a solution to a problem or opportunity. However, this does not explain where this motivation to innovate originated from. Previously in section 2.2, it was mentioned that innovation culture could positively change the project performance, and Dobni (2008, p. 541) had identified four dimensions that affect innovation culture. These dimensions were intention, infrastructure, influence, and environment. Finding these dimensions within the case studies can help explain the decision to start developing internal tools.

As previously stated in chapter three, the project members had previous experience developing internal tools at the CC. This previous experience suggests that at least some project members were engaged in the innovative imperative regarding internal tools. Also, the fact that the delivery project prepared to tackle the challenging specifications by including team members with a high-level of technical knowledge and practical development skills ready to develop these kinds of internal tools shows that the CC has adopted internal tools as part of its processes. Both of these factors are indications of *intent* for being innovative.

The case studies also make it clear that neither the project nor the CC set limits that prevented project members from bringing up new innovative solutions for problems they were facing. This factor is an indication of *infrastructure* that supports innovation. Furthermore, the fact that the team members developed both tools successfully during the project clearly indicates an *environment* suitable for innovations like internal tools.

In conclusion, both internal tools arose from the problems and opportunities encountered by the delivery project. More specifically, the problems brought up the need for action, and the opportunities made it possible for an internal tool to solve the problem. The problems which led to the demand for the tools varied in each situation, but the opportunities have some commonalities. Opportunities for repeatable or project enabling use cases paired with a relatively low development effort sparked the development of these valuable internal tools. Organizational culture might also affect the employee's willingness to create such tools. In the cases of these valuable tools, there seemed to be indications of multiple dimensions of innovation culture.

4.2 Reusability and scope of a new internal tool

The last section highlighted important attributes of internal tool projects by focusing on the initial spark for the development. One crucial but almost too obvious attribute to focus on still needs more attention: the tool's scope. After all, the scope can often seem to be determined by the situation at hand.

Every development project or project, in general, needs a scope. According to Hassan, Ahmad, and Zuhaira (2018, p. 208), *scope* specifies the contents and limits of a software project. Without any scope, there is no direction for the development. They also propose that it is usually complicated to define the scope accurately. They assert that this complexity is due to the massive amount of variables and information related to the project's subject and the project itself. In the case of internal tools, most of the variables and information needed can be derived straight from the project for which the internal tool is developed. These limits include, among other things, budgeting, scheduling, stakeholder, and human resource-related variables. Hassan *et al.* (2018, p. 208) categorize these limits as information related to the project itself. In the case of internal tools, this information comes pre-defined from the project for which the internal tool is developed.

However, some limits cannot be directly derived or are hard to derive from the project for which the internal tool is developed, even if they indirectly affect the previously mentioned pre-defined limits. These limits Hassan *et al.* (2018, p. 208) categorize as product or service-related information. In the case of software projects, they include the requirements for the software as the limits. As previously stated, according to Pohl *et al.* (2015, p. 8), software requirements can be divided into three categories: functional requirements, quality requirements, and constraints.

Even when requirements seem apparent, they do not appear from nowhere; they are always elicited. According to Pohl *et al.* (2015, p. 11–19), the knowledge for this elicitation comes from the requirements engineering. They define *requirements engineering* as a method for specifying and managing needs that are methodical and disciplined. The objectives are to know the necessary requirements, obtain stakeholder agreement on these requirements, document them according to established standards, and methodically manage them. Also, to minimize the risk of delivering a system that does not meet the stakeholders' needs, their desires and needs must be understood and documented. (Pohl *et al.*, 2015, p. 11–19.)

Requirements engineering unveils the system context. According to Pohl *et al.* (2015, p. 11–12), *system context* is a component of the system environment that is important for defining and comprehending the requirements of a system under development. They list people, systems in operation, processes, events, and documents as the main categories of the system context. People include the stakeholders, in the case of internal tools, the development team, and the project team for which the tool is created. Systems in operation can be any system that is connected to the new system. Processes are all of the processes that affect the new system or its development, including business processes. Events are physical or technical events that affect the system. Lastly, documents represent all documentation that affects the system or its development. These include previous system documentation, standards, and laws. On the other hand, *system boundary* is the line between the system alterable by the development process and the system environment that cannot be altered during the development. The system, system context, and their boundaries are presented in the Figure 7. (Pohl *et al.*, 2015, p. 11–14.)

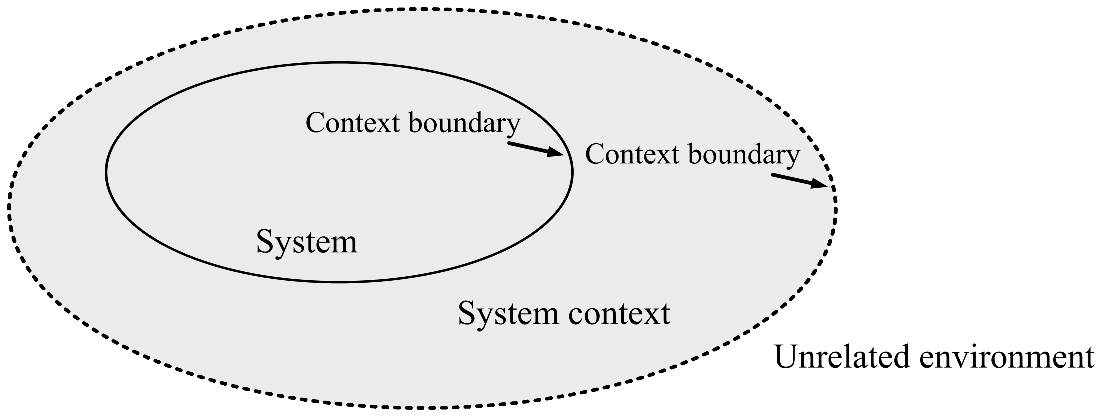

Figure 7. System, system context and their boundaries.

Note. Adapted from "Requirements Engineering Fundamentals" by Pohl and Rupp, 2015, p. 13.

In reality, the system and context boundaries are not as clear as Figure 7 implies. According to Pohl *et al.* (2015, p. 13–17), the system and context boundaries are usually defined more accurately only at the end of the requirements engineering process. During the requirements engineering, some of the interfaces and outputs of the system are not yet known. They call this undefined area between the system boundary and system context where the boundary moves *the grey zone*. Not only can the system boundary move due to new information arising during the requirements engineering process, but also the grey zone itself can move. This movement means that aspects that were not considered for modification initially might now be changed. This alteration can be caused by decisions made during the process or newly uncovered information. (Pohl *et al.*, 2015, p. 13–17.)

As internal tools are software, requirements engineering attributes and methods apply to them too. However, there are similarities in the Deployment Tool and the Migration Tool cases regarding these characteristics. In both cases, requirements engineering manifested as a collateral and continuous process instead of a rigid standalone phase. This was due to a large number of unknowns in the delivery projects for which the tools were developed. Also, a tight schedule meant that there was no time for an exhaustive self-contained phase where the requirements would be frozen. In both cases, the tools' development was not considered a standalone project but only part of the delivery project. This fact meant that only the necessary requirements were defined and only when they were to be implemented.

Another similarity for these development projects was the considerably broad and animate grey zone. It was known at the start of both development projects that there were multiple routes to the currently known goal but no clear path to follow. Also, it was known that there would be changes to the functionalities of both tools during the development. With the Deployment Tool, the needed core functionalities expanded during the development. Moreover, with the Migration Tool, the tool functioned as a platform for more minor use cases in addition to the significant core functionalities. In these cases, the system boundaries froze after the delivery project was almost completed.

Despite their similarities, both tools had very different scopes, both in their size and their content. The Deployment Tool was a much quicker and more agile response to the situation than the Migration Tool. For this reason, the core functionalities were also much more concise and more straightforward to implement. The environment for the original demand of Deployment Tool also directed the development to a high-level implementation from the start. This more general solution meant that the tool would support similar situations with fewer modifications in different internal projects than the Migration Tool. This applicability also paved the way for the Deployment Tool as an internal product.

The previous section established that the effort needed to create an internal tool is an important variable when considering the tool's value. As the scope varies widely from one tool to another and the scope mainly defines the effort needed to develop the internal tool, the effort needed for the tools varies widely. Ultimately, the delivery project will set the limits for the internal tool. In cases where the scope is left undefined, the risk of breaking these limits increases. One of these risks is *scope creep* which describes a situation where the effort used for the software may surpass the assigned budget if the scope of the tool increases during its development (Pratt, 2012, p. 152). For this reason, it is paramount that the presumed scope of the tool is known or at least investigated before starting the development and updated after changes to the scope.

4.3 Variables to consider before investing in internal tools

The cases presented in chapter three and their subsequent analysis in sections 4.1 and 4.2 have brought up many different attributes that significantly affected the success of these tools. These attributes are the repeatability or reusability of the tool, enablement provided by the tool, the effort required for the development of the tool, and the tool's scope. Although many other variables can affect the success of an internal tool, these were the foundation for it in these cases.

Repeatability or reusability of the tool describes the number of times supported use cases can be fulfilled. Repeatability can be examined in the scope of the tool's entire lifecycle or in the scope of the project for which the tool was initially developed. In some cases, the tool's entire lifecycle can also be limited to the initial delivery project. Usually, internal tools are not standalone projects but use the delivery project's budget. From the delivery project's point of view, reusability after the project is less important than during the project. For this reason, the focus on repeatability after the initial delivery project blurs the line between internal tools and internal products.

Enablement as an attribute describes the tool's ability to enable the project for which the tool was developed. In the strictest sense, enablement starts from the point where the project could not be completed within its limits without the tool. Enablement can also be viewed from a more broad perspective. This perspective would consider anything that helps the initial delivery project as enablement. As the word *tool* already implies that the tool enables some function, the more strict definition is used when describing enablement in the context of internal tools. Enablement and repeatability are not mutually exclusive. An internal tool can enable a delivery project while being reusable. However, as enablement and repeatability describe the benefits provided by the tool, one of them is required for the tool to be beneficial. These attributes can be considered as prererequisites for internal tool development.

The last two attributes, the effort needed for the tool's development and the tool's scope, are connected as previously discovered. The effort needed for finishing the tool's development is highly dependent on the tool's scope. Different internal tools can have very different use cases, forcing the effort to be estimated on a case-by-case basis. The delivery project can provide limits for the scope, but the actual functionality of the tool can be hard to specify as a pre-condition to development. Consequently, the confidence of effort estimates will vary. Lock (2013, p. 59–60) has provided a classification that demonstrates this range. They categorize estimations according to the confidence levels to ballpark, comparative, feasibility, and definite estimates. An estimate for a tool done in a hurry with only vague information would be a ballpark estimate. However, if there is no specific information about the tool but a similar project has been completed, this information can be enough for a comparative estimate. If the new tool's requirements are already well defined, the estimate would be a feasibility estimate. Due to the nature of internal tool development, definite estimates can only usually be done after most of the tool has already been developed.

The cases also revealed the conflict of resource usage with internal tool development; if the internal tool uses the same resources as the initial delivery project, all effort focused on the tool is removed from the actual delivery project. For this reason, developing an internal tool requires balancing between the pros and cons of each situation as a whole.

5 VALUE OF INTERNAL TOOLS IN SOFTWARE DELIVERY PROJECTS

Previous chapters have explored the benefits and challenges of internal tools. Based on these findings, it is still impossible to label internal tools as either beneficial or disadvantageous in a general sense. Based on the cases, it also seems that it is at least sometimes hard to specify the scope for the internal tools before starting the development. This chapter will help with this value generalization. The first section introduces a necessary concept for software delivery projects regarding their value, *time to value*. After that, the second section answers the final and the most critical research question by exploring how the value of internal tools can be estimated within a software delivery project.

Before delving deeper into the value of different concepts, the noun in itself needs to be clarified. Oxford English Dictionary (2021) defines *value* as "Worth or quality as measured by a standard of equivalence." Hence, the value of something needs to be measurable by an equivalence compared to something else's value. It is important to note that one standard of equivalences might not be convertible to other standards of equivalences. For this reason, the action of value estimation is heavily context-bound.

5.1 Time to value in software delivery projects

ICT projects have for a long time focused on speed in addition to the results of the project. One manifestation of this way of thinking is the concept of time to value. According to Hawkinson and Winter (1999), *time to value* is a time-focused method of measuring a project's success. To calculate time to value, organizations must assess how quickly a project could deliver some value to the organization, what that delivered value was, how it impacted the organization, and how it can be achieved again in the future (Hawkinson *et al.* 1999).

The importance of speed has only increased due to the continuing rise of SaaS services (Gartner, 2021a). As licensing is based on subscriptions in SaaS (Wohl *et al.*, 2010, 107), the recurring revenue from those subscriptions generally starts after the system is usable. This growing importance of time to value is especially true with delivery projects that are implementing SaaS software solutions. The longer the delivery project takes, the longer it takes for the recurring revenue to start (Wohl *et al.*, 2010, pp. 107–111). In the case of a delay, the recurring revenue missed is not delayed; it is lost. Even when speed has always been a benefit regarding software delivery projects, the growing significance of time to value in an organization's finances highlights the time aspect of software delivery projects.

5.2 Estimating the value of an internal tool in a software delivery project

At the start of this chapter, the general definition for the noun *value* was defined. It was also mentioned that value estimations are heavily context-bound. However, what equivalence standard can be used to define the value of internal tools in a software delivery project specifically, and how can it be measured? This question can be complex or straightforward, depending on how much is known. If the tool is finished and has served its purpose, the delivery project is complete, and the alternate estimation for the option without the tool is accurate, the tool's value can be calculated straight from the numbers.

The tool has value if the use of the internal tool is more financially beneficial than the alternative, including all its development costs. The more money the tool saved or helped to acquire, the better the value. On the other hand, if the tool is a worse option financially than the alternative manual work, it has no value. However, this simple calculation presumes that all of the effort and benefits regarding the internal tool are diligently booked, which is not usually the case. Even if the project is finished and all of the project-related benefits and costs are available, the manual alternative can be impossible to estimate accurately.

The more realistic scenario is where the value for the internal tool needs to be estimated before the full scope of the tool is known. This situation is encountered with every internal tool when the decision is made to move forward with development or not. Even when the complete picture might not be apparent at the time of that decision, some attributes can be estimated to support the decision. These attributes are the variables explored in the section 4.3; repeatability, enablement, the effort needed for the development, and the tool's scope.

The repeatability of the tool can be estimated based on the presumed use cases. If the tool has core use cases that are definitely required for the new internal tool, their recurrence can be estimated for the project. In the case of the Deployment Tool, the project team knew that the tool would be used on all following deployments in the project. Although it was unknown how many vaults would need to be deployed, the amount was still known to be high. These circumstances were the same with the Migration Tool.

The same basic principle applies to enablement as does to repeatability. When a delivery project necessitates certain use cases, including them will increase the tools enablement. The amount of enablement the tool provides depends on the situation it is enabling. If the tool makes the whole delivery project possible while it would not be profitable without the tool, the enablement is undoubtedly high.

As the tool's scope and effort needed to develop it are connected, either the preliminary scope can be used to estimate the effort, or the effort can be estimated as-is. The estimation does not necessarily need to be exceedingly accurate to estimate the value of a new internal tool. However, it is important to be careful not to underestimate the effort required. In the cases of the Deployment Tool and Migration Tool, the initial estimation for the effort was done by the experienced developer.

These variables by themselves cannot indicate whether a tool would be of great value. However, combining these variables can lead to an excellent estimation of the value of an internal tool for a specific delivery project. The repeatability and enablement represent the beneficial attributes that ultimately make the internal tool valuable for the delivery project. The effort required for the development, on the other hand, is the counterbalance to the benefits. To reap the benefits of the tool, you must accept the cost of the effort required to develop it.

An internal tool can have a high value for the delivery project when the tool's functions can be reused multiple times or when the tool acts as an essential enabler in the delivery project. However, high value requires that the effort for the development is reasonably low compared to the benefits. An internal tool that requires high effort can be high value, but the larger subproject can cause problems for the delivery project itself. For this reason, high-value yet high-effort tools usually fall in the category of internal products.

Internal tools that have low repeatability and enablement but high effort undoubtedly have little or no value to the delivery project. However, internal tools with low repeatability, low enablement, and a small amount of effort are much more challenging to generalize. From the delivery projects' point of view, these internal tools have a low potential to affect the project.

Described generalizations can also be presented on a matrix. The top-left and bottom-right corners indicate the clear cases where an internal tool should or should not be developed. The space between these areas, including the bottom-left and top-right corners, are cases where the estimation should be done on a case-by-case basis. This internal tool value estimation matrix is presented in Figure 8.

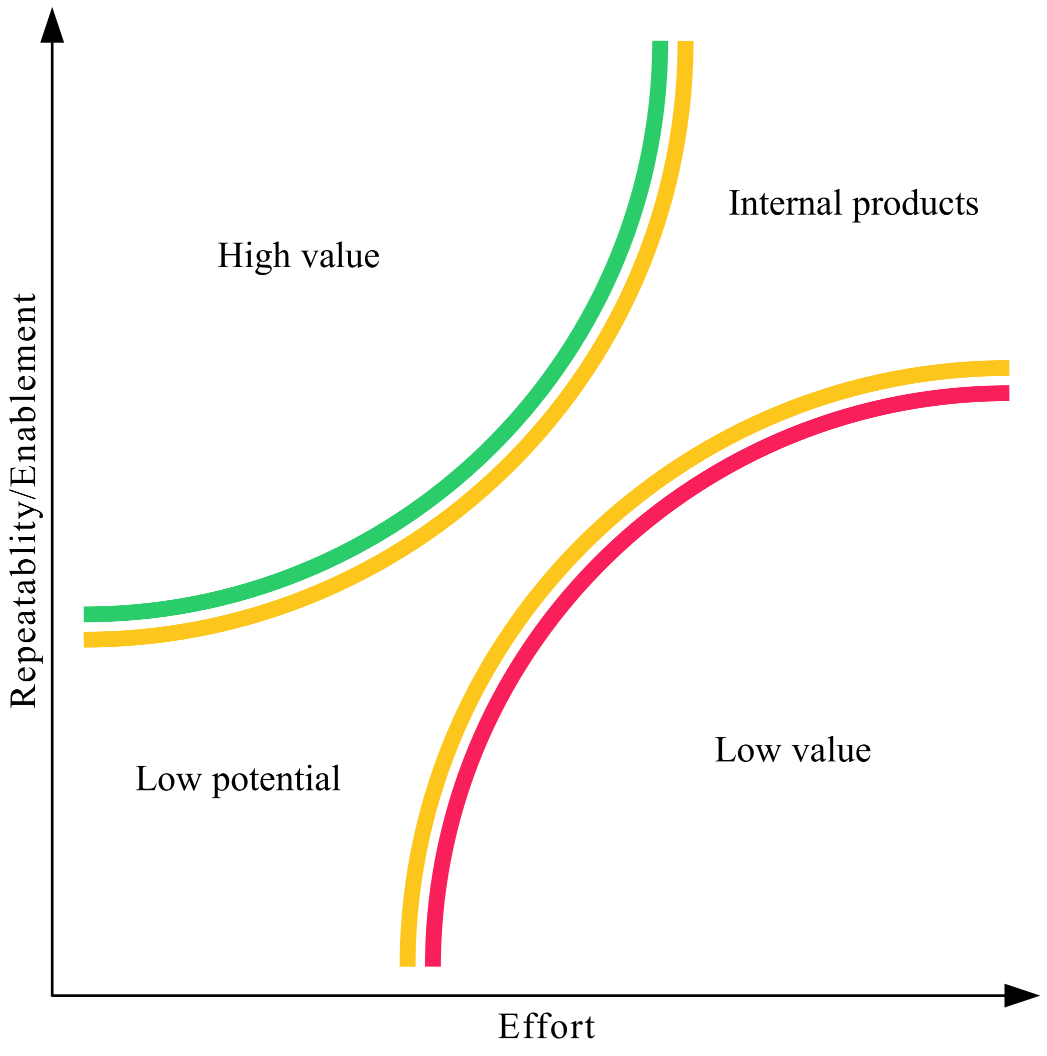

Figure 8. Internal tool value estimation matrix.

The lack of specific values for boundaries where the tool turns from high-value to the middle section or middle section to low-value is intentional. Each case must be considered in the context of the delivery project. For example, both the Deployment Tool and Migration Tool required considerable effort. However, both tools were used multiple times during the delivery project, resulting in significant time savings. Also, both tools were essential parts of the delivery project. Without them, the delivery project would not have succeeded. Both tools saved and generated considerably more resources for the delivery project than used. In any case, the effort should never surpass the resource budget of the whole delivery project.

It should be noted that the estimations based on these generalizations are only as accurate as the information used in the estimation. High-value estimation based on reliable, detailed information will most likely result in a high-value tool like the Deployment Tool or Migration Tool. However, overestimating the repeatability or the enablement provided by the tool as well as underestimating the effort needed for the development will lead to an underwhelming tool. Also, tools that fall into the middle section based on the preliminary information are a red flag for the delivery project. Internal tools are usually built around the core use cases. If the preliminary information for the estimation is already indicating low benefits compared to the presumed effort, the tool might not be worthwhile.

6 CONCLUSIONS

The objectives of this research were to define internal tools, examine prerequisites and effort required for internal tool development, and investigate their value to software delivery projects. The first research question is: What are internal tools in software delivery projects? In general, even if an idea for a tool is good, an idea in itself is not enough. An internal tool is an innovation. Therefore, the idea needs to be also implementable and functional. The literature review carried out to answer the first research question did not find an established definition for *internal tools*, so a new definition was proposed. In the context of software tools, internal tools are auxiliary tools developed by employees within an organization for specific internal use cases, not intended for productized use. Unmodified third-party tools and open-source projects, simple derivatives of third-party tools or open-source projects, and software languages, libraries, frameworks, or protocols are not internal tools. However, the distinction between internal tools and internal products can be vague. Internal tools can be developed into widely used internal products. For this reason, some auxiliary tools can be both during their lifecycles.

The second research question is: What are the prerequisites for internal tool development, and what type of effort does their development take? These questions were answered with reference to applicable literature and the facts from presented case studies. The included case studies revealed that internal tools can have much variance, but there are many commonalities. The benefits of internal tools are derived from repetitive use cases or use cases that enable the delivery project. Having some of these benefits is also a prerequisite for internal tool development. As the counterbalance for the benefits of internal tools, there is the development effort. Internal tools are not usually developed as standalone projects. For this reason, the delivery project will provide limits for the internal tools that can be developed for it. Furthermore, the tools' scope will determine the effort the development takes. As the scope varies widely between different tools, it is paramount to investigate the scope before starting the development.

The third and the main research question is: Does internal tool development bring value to software delivery projects, and how can that value be measured? The answers to these questions were provided in the conclusions for the first and second research questions together with the presented case studies. To determine the value of an internal tool to a delivery project, it needs to be measurable. This calculation is easy if the tool's effect on the project's resources can be quantified accurately. When the tool has saved the project's resources, or generated more of them, considering the resources used for the development of the tool, it has value to the project. On the other hand, if the alternative option has used fewer resources such as work hours, time, or money, the tool has no value for the project.

The problem is that this calculation is often impossible based on the information available. Either information about the tool is lacking, or the alternative to the tool is only a vague estimation. For this reason, the value of an internal tool needs to be estimated separately. For the project team to do this estimation, the benefits and disadvantages need to be weighed against each other to determine the value of an internal tool to software delivery project.

The internal tool value estimation matrix was proposed to help with the value estimation of an internal tool. This matrix helps to separate high-value tools from low-value cases where development is not worthwhile. High-value tools have a high combination of repeatability and enablement compared to the effort needed for the development. Low-value tools are the exact opposite; their benefits are small compared to the effort needed for the development. Between high and low value is a middle ground that indicates the possible problems with the proposed tool. This middle ground does not mean the tool cannot be developed; it only recommends that it be estimated with greater specificity on a case-by-case basis.

Still, it must be recognized that value estimations based on the internal tool value estimation matrix depend heavily on the available data. If the required effort is underestimated while overestimating the potential for repeatability or enablement, the matrix can give an overly optimistic valuation for the tool. The same applies vice versa. If the estimation does not distinctly indicate high or low value, the benefits and scope of the tool need to be specified further.

To conclude, internal tools can bring significant value to software delivery projects despite their dynamic development process. To ensure the tool brings value to the delivery project, estimation of the tool's value should be performed before the start of the development. Early estimation can bring out aspects of the tool that were not yet considered and save the project resources.

7 DISCUSSIONS

Although this report focuses on internal tools in a software delivery project, these results could apply generally to software-based internal tools. At least software projects that value time savings with general resource savings would presumably value internal tool development similarly to delivery projects. However, different software project types can have attributes that change how internal tools create value. Delivery or deployment projects have an emphasized connection to the time aspect, as previously discussed. Delivery projects also have a start and an end. This relatively small time window forces the auxiliary tool development to be fast and dynamic. After all, the tool needs to return sufficient value during the delivery project. Continuous processes or projects seeking more gradual advancements might opt for internal product development instead of internal tools. For this reason, future research could focus on the value of internal auxiliary tools for different project types or processes.

Future research should also be conducted on atypical cases or cases that proved to be low-value tools despite being estimated to be high-value. The case studies presented in this work represent internal tools for a software delivery project where the value was estimated to be high, and the tools subsequently proved to be of high-value. Examining internal tools estimated to be higher than the resulting tool's value could provide valuable information regarding internal tool value estimation.

The explored cases also found indications of multiple dimensions of innovation culture. Future research into the connection between innovation culture and internal tools could reveal whether innovation culture is a prerequisite for internal tool development or not. Even if innovation culture would not act as a prerequisite for internal tool development, the number of valuable internal tools created in an organization could correlate with its innovation culture. After all, the innovation process for internal tools depends heavily on the individuals' willingness to promote their new idea.

It should also be noted that the amount of data for the empirical research is modest due to the lack of information about previously developed internal tools. Even when the information for these cases was, as previously mentioned, combined from multiple sources such as project documentation, reports, and interviews. The best case would be to follow multiple projects that would develop internal tools from start to finish. However, this is difficult because not every project will need internal tools, and those projects that do can take a considerable amount of time. These case studies provide a great understanding of the general mechanisms of value creation of internal tools in delivery projects. Still, possible future quantitative research focused on the measurable effects of internal tools could reveal a more granular scale for evaluating internal tools.

REFERENCES

Abbey, A. and Dickson, J. (1983) 'R&D Work Climate and Innovation in Semiconductors', *Academy of Management journal*, 26(2), pp. 362–368.

Ahonen, J. and Savolainen, P. (2010) 'Software engineering projects may fail before they are started: Post-mortem analysis of five cancelled projects', *The Journal of systems and software*, 83(11), pp. 2175–2187.

Amabile *et al.* (1996) 'The social psychology of creativity: A componential conceptualization', *The Academy of Management Journal*, 39(5), pp. 1154–1184.

Amabile, T. (1983) 'The social psychology of creativity: A componential conceptualization', *Journal of Personality and Social Psychology*, 45(2), 357–376.

Ansorena, M., del Valle, C. and Salvadori, V. (2010) 'Application of Transfer Functions to Canned Tuna Fish Thermal Processing', *Food science and technology international*, 16(1), pp. 43–51.

Baregheh, A., Rowley, J. and Sambrook, S. (2009) 'Towards a multidisciplinary definition of innovation', *Management decision*, 47(8), pp. 1323–1339.

Bonver, E. (2008) 'Security Testing of Internal Tools', *IEEE security & privacy*, 6(1), pp. 81–83.

Brown, T. and Katz, B. (2011) 'Change by Design', *The Journal of product innovation management*, 28(3), pp. 381–383.

Büschgens, T., Bausch, A. and Balkin, D. (2013) 'Organizational Culture and Innovation: A Meta-Analytic Review', *The Journal of product innovation management*, 30(4), pp. 763–781.

Cambridge Dictionary (2021) *Investment*. Available at: https://dictionary.cambridge.org/dictionary/english/investment (Accessed: 23 October 2021).

Chandler, G., Keller, C. and Lyon, D. (2000) 'Unraveling the Determinants and Consequences of an Inn ovation-Supportive Organizational Culture', *Entrepreneurship theory and practice*, 25(1), p. 59.

Chesbrough, H. and Euchner, J. (2011) 'Open Services Innovation: An Interview with Henry Chesbrough', *Research technology management*, 54(2), pp. 12–17.

Cowan, R. (1998) 'Product globalization', *International business (Rye, N.Y.)*, 11(3), p. 19.

Crouch, A. and Duerden, R. (1995) 'Delay-path strategy has test payback', *Electronic engineering times*, (870), p. 42.

Crowe et al. (2011) 'The case study approach', *BMC Medical Research Methodology*, 11(1), p. 100.

Damanpour, F. (1996) 'Organizational Complexity and Innovation: Developing and Testing Multiple Contingency Models', *Management science*, 42(5), pp. 693–716.

Dobni, C. (2008) 'Measuring innovation culture in organizations: The development of a generalized innovation culture construct using exploratory factor analysis', *European journal of innovation management*, 11(4), pp. 539–559.

Eurostat (2021) Labour cost levels by NACE Rev. 2 activity. Available at: https://ec.europa.eu/eurostat/databrowser/view/lc_lci_lev/default/table?lang=en (Accessed: 5 September 2021).

Feagin, J., Orum, A. and Sjoberg, G. (1991) A Case for the Case Study. Chapel Hill: The University of North Carolina Press.

Garcia, K. (2021) *The state of internal tools in 2021*. Available at: https://retool.com/blog/state-of-internal-tools-2021/ (Accessed: 20 September 2021).

Gartner (2021a) Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 23% in 2021. Available at: https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021 (Accessed: 3 August 2021).

Gartner (2021b) Gartner Glossary: Metadata. Available at: https://www.gartner.com/en/information-technology/glossary/metadata (Accessed: 19 October 2021).

Hassan, I., Ahmad, N. and Zuhaira, B. (2018) 'Calculating completeness of software project scope definition', Information and software technology, 94, pp. 208–233. doi: http://dx.doi.org/10.1016/j.infsof.2017.10.010

Hawkinson, A. and Winter, B. (1999) 'Optimizing Time-to-Value', Intelligent enterprise (San Mateo, Calif.), 2(14), p. 60.

Hughes et al. (2018) 'Leadership, creativity, and innovation: A critical review and practical recommendations', The Leadership quarterly, 29(5), pp. 549–569.

Internal (2021) *Homepage*. Available at: https://www.internal.io/ (Accessed: 13 September Internal).

Johnston, J. (2021) *What are internal tools? The definitive guide to internal tools in 2021*. Available at: https://www.budibase.com/internal-tools/ (Accessed: 20 September 2021).

Lewis, B. and Kaiser, D. (2019) *There's no such thing as an IT project: a handbook for intentional business change*. 1st edn. Oakland, CA: Berrett-Koehler Publishers, Inc.

Ling, X., Gautam, R. and Vallabh, S. (2012). 'Efficiency or Innovation: How Do Industry Environments Moderate the Effects of Firms' IT Asset Portfolios?', *MIS Quarterly.* 36(2), pp. 509–528

Lock, D. (2013) *Project management.* 10th edn. Burlington, Vt: Gower.

Mills, A., Durepos, G. and Wiebe, E. (2010) *Encyclopedia of case study research*. Los Angeles, California: SAGE Publications.

Nambisan, S. and Nambisan, P. (2008) 'How to profit from a better 'virtual customer environment'', *MIT Sloan management review*, 49(3), p. 53.

OECD (2005) *Product Innovation*. Available at: https://stats.oecd.org/glossary/detail.asp?ID=6868 (Accessed: 21 September 2021).

Orum, A. (2015) 'Case Study: Logic'. doi: https://doi.org/10.1016/B978-0-08-097086-8.44002-X

Owens, T. and Fernandez, O. (2014) The lean enterprise : how corporations can innovate like startups. Hoboken, New Jersey: Wiley.

Oxford English Dictionary (2021) *Value, n.*. Available at: https://www-oed-com.ezproxy.cc.lut.fi/view/Entry/221253?rskey=2guClq&result=1#eid (Accessed: 16 November 2021).

Pohl, K. and Rupp, C. (2015) *Requirements Engineering Fundamentals*. 3rd edn. Springer International Publishing AG.

Pratt, D. (2012) *The IT project management answer book.* 1st edn. Tysons Corner, Virginia: Management Concepts Press.

Pressman, R. (2001) *Software Engineering: A Practitioner's Approach*. 5th edn. New York, NY: McGraw-Hill Education.

Pressman, R. and Maxim, B. (2015) *Software Engineering: A Practitioner's Approach*. 8th edn. New York: McGraw-Hill Education.

Sherwood, A. and Covin, J. (2008) 'Knowledge acquisition in university-industry alliances: An empirical investigation from a learning theory perspective', *The Journal of product innovation management*, 25(2), pp. 162–179.

Stake, R. (1995) *The Art of Case Study Research*. London: SAGE Publications.

Steffora, A. (1998) 'An EDA Tool Option', *Electronic news (1991)*, 44(2221), p. 42.

Teresko, J. (2004) 'Open innovation? Rewards and challenges; the rewards come from connecting internal product development to outside technologies. But an open innovation strategy also can bring unexpected corporate challenges including cultural adjustments', *Industry week*, 253(6), p. 20.

Trott, P. (2012) *Innovation management and new product development.* 5th edn. Harlow: Financial Times/Prentice Hall.

Wallin, J. (2012) 'Turning Internal Product Knowledge into External Service Offers: Building PSS Capabilities', *The Philosopher's Stone for Sustainability*, pp. 327–332. doi: https://doi.org/10.1007/978-3-642-32847-3

Wohl, A. and Simon, P. (2010). *The Next Wave of Technologies : Opportunities in Chaos*. Hoboken, NJ, USA: John Wiley & Sons.

Yin, R. (2003) *Case Study Research : Design and Methods*. 3rd. Thousand Oaks, California: SAGE Publications.

Zhang, S., Tang, T. and Wu, F. (2021) 'The ambidextrous patterns for managing technological and marketing innovation', *Industrial marketing management*, 92, pp. 34–44.