Enterprise Software as Design Infrastructure

Magnus Li

Thesis submitted in partial fulfilment of the requirements

for the degree of Philosophiae Doctor (PhD)

Department of Informatics

Faculty of Mathematics and Natural Sciences

University of Oslo

March 2022

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2531

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Graphics Center, University of Oslo.

In Memory of Arne and Ingeborg Li

(1928 – 2016, 1925 – 2021)

Table of Contents

Abstract

Information systems (IS) literature has a tradition of emphasizing the benefits of the contextual design of information technology (IT). This means designing technology features based on the practices and needs of a group of end users in a particular context to secure their usability and relevance. However, generic enterprise software (ES) solutions can be found at the center of many digitalization projects in public and private organizations. Instead of being based on a particular context, ES solutions are designed to accommodate the needs of a diverse set of user organizations. A persistent challenge discussed in IS literature is that these ES solutions often fail to offer a sufficient fit with idiosyncratic practices and needs, limiting the user organizations' benefits of adoption. Studying ES design is also challenging as it is often distributed across time and space, as well as among several actors and constituencies, such as one or several vendors, partners specializing in implementation, and user organizations. Consequently, existing literature offers partial and fragmented accounts of ES design by focusing on single locales, short timeframes, or high-level analyses of the relations among different actors. Primarily, the focus is on the challenges of implementing ES in individual user organizations, how vendors work to align the needs of different user organizations, or how they manage an ecosystem of actors around their ES solutions. How ES can be designed as generic solutions, while accommodating a diverse set of user organizations' specific needs, remains an open and relevant question for IS research and practice.

Based on an engaged scholarship research project involving collaboration with an ES vendor and a group of partners specializing in implementation, this thesis extends existing knowledge on ES design by examining two related research questions (RQs):

RQ 1. How can ES be designed to accommodate the specific needs of a diverse set of user organizations?

RQ 2. How can researchers study ES design in collaboration with relevant practitioners?

Addressing RQ 1, the thesis develops a theoretical framework to understand ES design. The framework conceptualizes two types of design processes – *generic-level* and *implementation-level design* – as well as the set of generic software features and knowledge resources linking the two as a *design infrastructure*. The framework highlights how generic-level and implementation-level design work in tandem to accommodate various user organizations and points to ES implementation as a potential setting for contextual design.

The thesis then examines the challenges of and conditions for designing ES according to particular user organizations' needs during implementation-level design. It identifies how implementation projects are configured, the partners' practices of implementing

ES for user organizations, and the design infrastructure as the conditions for contextual implementation-level design. The thesis finds that the design infrastructure exerts a formative effect on the partners' implementation practices and on how implementation projects are configured.

Furthermore, the thesis examines how a platform strategy (i.e., engaging partners in designing, maintaining, and distributing generic add-on modules or apps that extends the features of the solution offered by the vendor) affects the dynamics of ES design. The thesis identifies a set of challenges and associated considerations for ES vendors.

In response to RQ 1, the thesis suggests for ES vendors to see their primary work as to cultivate their ES solutions as design infrastructures supporting contextual implementation-level design. From the theoretical framework and empirical findings, the thesis outlines and discusses concrete implications for IS research and practice related to ES design.

For RQ 2, the thesis conceptualizes a *design lab* as an approach to studying ES design based on the theoretical framework and using engaged scholarship as a methodological basis. The thesis shows how such an approach can involve diagnostic, design, and intervention-oriented research on generic-level and implementation-level design processes, as well as the resources of the design infrastructure. This is achieved by collaborating with practitioners through various forms of engaged scholarship, such as case studies, action research, and design science research. Different forms are selected based on the evolving understanding of the problem and potentials for collaboration with practitioners. With this, the thesis offers a methodological contribution to IS conversations concerning ES design in particular and engaged scholarship in general.

Preface

This dissertation consists of four papers and an introductory section. The papers are listed below and included as appendices.

1. **Making Usable Generic Software. A Matter of Global or Local Design?**

   Magnus Li and Petter Nielsen, Scandinavian Conference on Information Systems (2019)

2. **Generic Enterprise Software Implementation as Context for User-Oriented Design: Three Conditions and their Implications for Vendors**

   Magnus Li, Scandinavian Conference on Information Systems (2021)

3. **Organizing Enterprise Software Ecosystems for Design: Considerations for Adopting a Platform Strategy**

   Magnus Li and Petter Nielsen, to be submitted to an international IS journal

4. **Entering and Organizing Engaged Information Systems Research**

   Magnus Li, to be submitted to an international IS journal

Acknowledgements

My PhD journey has been interesting and full of learning. I have greatly enjoyed being part of the generous environment of the information systems research group and the HISP. While there are many people to thank for making the last four years such a great adventure, I use this opportunity to mention a few of them.

First and foremost, I am deeply grateful to my primary supervisor Petter Nielsen, for encouraging and supporting me in embarking on a PhD degree in the first place, for facilitating both my professional growth as a researcher and the development of the design lab in every possible way, and for helping me sharpen my thoughts and writing throughout the process. Your significance to me and the master's students involved in the design lab cannot be overstated. I express my appreciation to my second supervisor Suhas Govind Joshi, for bringing me into teaching during my undergraduate studies, and since then, for encouraging me to continue with both teaching and research at the Department of Informatics. Your comments and open-ended questions, as well as our discussions on concrete papers, this thesis, and other matters are always deeply engaging, generative, and valuable. I also thank Kristin Braa; similar to Petter, she has been supportive of me and the design lab in every way. I have learned much from the qualities of Petter, Joshi, and Kristin that I hope to bring with me throughout my career and life.

Second, I am indebted to my friends and colleagues who have helped me by commenting on my manuscripts and participating in fruitful discussions related to my research. Particularly deserving of mention are Eric Monteiro, Miria Grisot, Sundeep Sahay, Egil Øvrelid, Alexander Kempton, Jens Johan Kaasbøll, Brian Nicholson, Jørn Braa, Terje Aksel Sanner, Silvia Masiero, and Johan Ivar Sæbø.

Third, I acknowledge the important work of the master's students who have been or are still part of the design lab and thank you for our many great discussions on both academic and unrelated matters.

Fourth, I extend my gratitude to the members of the DHIS2 core team and the partners in India, Mozambique, Malawi, Tanzania, Sri Lanka, the US, and Uganda, particularly Ola, Austin, Scott, Jyotsna, Zeferino, Emilio, Tiwonge, Mahoundi, and many more.

Finally, I thank my family, friends, and Vilde in particular for always being supportive of me in all aspects of my life.

Magnus Li
Oslo, 2022

1. Introduction

In this thesis, I examine how generic enterprise software (ES) can be designed to accommodate the specific needs of a diverse set of user organizations and how researchers can study ES design in collaboration with relevant practitioners. Many public and private organizations are in the midst of extensive digitalization efforts, attempting to leverage opportunities afforded by innovations in information technology (IT). As organizations, at some level, are always unique in their practices and needs, the field of information systems (IS) has a long tradition of emphasizing the relevance of the contextual design of IT. In this thesis, I employ the term *contextual design* when referring to agile and user-oriented approaches for designing technology features based on a group of end users' particular practices and needs. A common argument for such approaches is that to attain success in IT implementation, there must be an 'artful' integration (Suchman, 2002) between organizational practices and technology design, requiring an emphasis on the particularities of the context of use (Baxter & Sommerville, 2011). However, at the center of many digitalization efforts, we find generic enterprise software (ES) solutions, such as enterprise resources planning (ERP), supply chain management, and health management solutions. These are implemented by user organizations to "fulfill a broad range of essential organizational information processing needs on an organization-wide scope" (Haines, 2009, p. 183). As the qualifying term *generic* indicates, these solutions are not designed with a specific context in mind but are intended to serve a diverse set of user organizations. For instance, solutions from ES vendors, such as SAP, Oracle, and Salesforce, are widely adopted by organizations to serve their diverse sets of needs.

Aiming to be relevant to *all*, it is neither desirable nor possible for ES vendors to accommodate each user organization's particular practices and needs (Pollock et al., 2007; Sia & Soh, 2007). Rather, the aim is to benefit from economies of scale by sharing design, development, and maintenance costs for the same software features among several adopting user organizations (Haines, 2009). Accordingly, the process of implementing generic ES solutions in user organizations has traditionally been portrayed in IS literature as somewhat the opposite of contextual design, that is, as one of adapting the users' practice according to the generic software (Berente et al., 2016; Kallinikos, 2004a). Argued as a consequence of their generic nature and limited flexibility, a persistent challenge discussed in IS literature is that ES solutions often fail to offer a sufficient fit with idiosyncratic practices and needs, thus limiting the user organizations' benefits gained from implementation (Davenport, 1998; Sia & Soh, 2007; Strong & Volkoff, 2010). The challenges arising from the solutions' limited ability to accommodate specific needs are argued to have adverse consequences for end users and user organizations alike by decreasing worker satisfaction, undermining the

organizations' competitive advantage, and reducing their overall performance (Berente et al., 2016; Tan et al., 2020). For instance, figures from 2016 indicated that 57% of ERP projects exceeded their budgets, while only 46% delivered benefits as expected (Tan et al., 2020). The challenges and the relative failures are often attributed to misalignments between the generic solutions and individual user organizations' needs (Berente & Yoo, 2012; Soh & Sia, 2008; Strong & Volkoff, 2010).

Although ES is a much examined and discussed phenomenon, the current understanding of ES design and of the ability of ES to accommodate specific needs remains partial and fragmented in IS literature. Three dominant perspectives co-exist. First, the majority of the studies examine the challenges and considerations that emerge during ES implementation from the perspective of the user organization (Berente et al., 2016; Markus & Tanis, 2000; Strong & Volkoff, 2010; Tan et al., 2020). Second, a smaller body of the literature examines how vendors work to design generic ES solutions by curbing the diverse needs of user organizations to offer highly standardized solutions that are "ready to travel" between contexts (Gizaw et al., 2017; Pollock et al., 2007; Kallinikos, 2009, p. 915). The third perspective focuses on how ES are often designed within ecosystems of multiple actors (Dittrich, 2014; Sarker et al., 2012); it also shows that ES vendors increasingly pursue platform strategies to support and encourage design and innovation beyond their organizational boundaries (Foerderer et al., 2019; Magnusson & Nilsson, 2013). While the three perspectives shed light on different relevant elements of ES design, they offer limited and fragmented accounts of and advice on how ES vendors may address the persistently relevant challenge of designing ES to accommodate the specific needs of a diverse set of user organizations.

In this thesis, I seek to add to the knowledge of ES design. Based on a four-year engaged scholarship research project, I have explored the challenges of designing ES to accommodate a diverse set of user organizations empirically, following a generic ES called DHIS2. DHIS2 is used by public and private organizations in more than 80 countries, mainly in Asia, Africa, and South America. It is primarily implemented to support the collection, analysis, and presentation of health management information. DHIS2 is primarily designed and maintained by a vendor at the University of Oslo, Norway, and implemented by user organizations with the support of partner organizations that specialize in DHIS2 implementation. Although with a relatively close relation to the vendor, the partners are independent consultancy organizations established in a number of countries, such as India, South Africa, Vietnam, Tanzania, Rwanda, Mozambique, and Uganda.

As DHIS2 is subject to adoption by many user organizations, it is a challenge to design software that accommodates the needs of all. For instance, there are variations in the optimal ways to structure workflows in user interfaces (UIs), to appropriate terminology, and what functionality is and may emerge as useful in a particular context.

This challenge is relevant for several actors. For the ES vendor in Norway, it is important to provide generic software features that sufficiently support existing user organizations while attracting new ones. However, as needs may be incompatible and to avoid the accumulation of too many features that are only relevant to a few, it is not feasible to accommodate needs specific to each and to do so in an agile and timely manner. For user organizations, it is vital that the ES sufficiently serves their needs. At the same time, the costs of custom development work and maintenance of custom features over time are ideally avoided or kept to a minimum. Accordingly, the partners specializing in implementing DHIS2 for user organizations want to offer cost-effective solutions by relying on generic software features. However, they often face user needs that cannot be accommodated with the generic features. To satisfy the user organizations, partners must respond rapidly to specific and evolving needs, but this may be challenging to do with the generic software features provided by the vendor. In these situations, either the specific user needs must be dismissed or the user organization must invest in costly development and maintenance of custom features. Thus, designing DHIS2 to accommodate its diverse set of user organizations represents a key challenge for multiple actors, a challenge shared with many ES vendors, partners, and user organizations in general (Haines, 2009; Pollock et al., 2007; Tan et al., 2020).

1.1 Research Questions (RQs)

In this thesis, I address two related research questions (RQs):

RQ 1. How can ES be designed to accommodate the specific needs of a diverse set of user organizations?

I address RQ 1 and contribute to IS research and practice related to ES design by doing three things:

First, I develop a theoretical framework to understand ES design. Whereas ES design is argued to be a collective effort among multiple actors spanning different timeframes and levels (Dittrich, 2014; Koch, 2007; Williams & Pollock, 2012), most conceptualizations of ES design focus on a single place or perspective, such as implementation in individual user organizations (Berente et al., 2016; Markus & Tanis, 2000; Strong & Volkoff, 2010), or the vendors' design processes (Gregory, 2014; Pollock et al., 2007). Building on existing insights from the literature and on the empirical findings from the case of DHIS2, the framework developed in this thesis seeks to capture the overall dynamics of ES design. To do so, the framework features two types of design processes: *generic-level* and *implementation-level design*. Generic-level design concerns designing and maintaining generic software features that are part of a *design infrastructure* that supports implementation-level design processes in configuring and extending the generic features according to specific user needs. The

framework helps analyze the dynamics between the design of generic software features and the processes of configuring and extending these during implementation in specific user organizations. It further highlights how generic-level and implementation-level design work in tandem to accommodate a diverse set of user organizations and points to implementation-level design as a potential context for contextual design.

Second, I examine the practices and challenges of conducting contextual implementation-level design. Existing studies of ES implementation almost exclusively "focus on organizational and social dynamics" within individual user organizations "while the role and impact of technology goes largely untheorized" (Berente et al., 2019, p. 26). Some researchers also examine various approaches to customizing generic software features during ES implementation (Haines, 2009; Singh & Pekkola, 2021), but their usual advice is to avoid customization (Rothenberger & Srite, 2009). Meanwhile, the literature on contextual design predominantly assumes a bespoke software development context, which differs significantly from that of ES implementation (Dittrich, 2014; Edwards et al., 2010; Sommerville, 2008). However, today's rapidly changing business environment and innovations in technological possibilities suggest that ES must accommodate flexibility and agility for individual user organizations (Elragal et al., 2020; Tan et al., 2020). Responding to this demand, several ES vendors advertise their solutions as compatible with and supportive of contextual design during implementation. For instance, SAP promotes the use of design thinking as a method intended to support contextual design and to help innovate during the implementation of their solutions (Johnson, 2018). Accordingly and acknowledged by others as well (e.g., Dittrich, 2014; Sommerville, 2008), the context of implementation seems to potentially play a larger role in ES design than portrayed by the perspectives in existing IS literature.

This thesis complements existing perspectives with my examination of implementation as a setting for contextual design. My primary focus is on how the DHIS2 partners play an essential role in designing software according to user organizations' needs by leveraging the design infrastructure's generic software features. By studying the practices and challenges among the partners and their implementation projects, I identify key conditions affecting the potential for contextual implementation-level design: the implementation practices of partners, how implementation projects are configured in terms of scope and structure, and the generic software features and their adaptation capabilities (i.e., the design infrastructure). Furthermore, I find that the design infrastructure exerts a formative effect on the implementation practices of the partners, on how implementation projects are configured, and thus, on the process of implementation-level design. I argue that this formative effect has important implications for ES vendors that seek to accommodate a diverse set of user organizations better.

Third, I investigate how adopting a platform strategy, that is, engaging the partners in designing, maintaining, and distributing generic add-on modules or apps that extend the features of the solution offered by the vendor, affects the dynamics of software design. Based on this, I identify a set of challenges and associated considerations for ES vendors. IS literature has recently taken an interest in how several prominent ES vendors adopt platform strategies seeking to move the locus of product design and innovation beyond their own organizational boundaries (Bender, 2021; Foerderer et al., 2019; Wareham et al., 2014). Rather than being the sole provider of generic software features, the vendor attempts to leverage the partners' expertise, resources, and intimate knowledge of users' needs, and support and encourage the partners to design and distribute apps. This is suggested to help address the challenge of accommodating a diverse set of user organizations by expanding the portfolio of generic software features (Bender, 2021).

In the case of DHIS2, the vendor is currently attempting to pursue a platform strategy to support and encourage partners to design, maintain, and distribute generic software features for use across user organizations. With a detailed analysis of how implementation-level design is affected by adopting a platform strategy, I identify several challenges to platform strategies, particularly in the ES context. From these challenges and the vendor's reflections on current and future measures to address them, I identify important considerations for ES vendors that adopt a platform strategy to help them better accommodate a diverse set of user organizations. The considerations are 1) how to partition design and maintenance between actors and processes to best provide flexibility to address specific user needs, while sharing most of the development and maintenance costs between implementations, 2) how to address uncertainties tied to the continuity and future direction of the design of generic software features offered by partners, and 3) how to support partners in remaining cognizant of and navigating an evolving set of heterogeneous generic software features provided by both the platform owner and the partners.

In short, regarding RQ 1, my findings suggest that for ES to accommodate the specific needs of a diverse set of user organizations, ES vendors should seek to cultivate their ES solutions as design infrastructures supporting contextual implementation-level design. I elaborate on the challenges, opportunities, and considerations for following such a route. Furthermore, I discuss my findings' relevance and implications for IS research and offer a set of concrete implications for practice related to ES design, involving ES vendors, partners specializing in ES implementation, and user organizations.

RQ 2. How can researchers study ES design in collaboration with practitioners?

For RQ 2, my thesis offers two methodological contributions aimed at both the IS literature concerned with the study of ES design and the broader literature on engaged scholarship in IS research.

First, closely related to the contributions regarding RQ 1 and the theoretical framework, my thesis offers a conceptualization of a design lab as an approach to studying ES design in collaboration with relevant practitioners. The challenges of studying ES design processes pose a major obstacle to developing knowledge in this area. ES is subject to "shaping […] distributed in time and space" (Williams & Pollock, 2012, p. 3), potentially by an ecosystem of multiple actors operating in different constituencies (Dittrich, 2014; Koch, 2007). However, the majority of existing studies are limited by focusing on single locales, primarily the level of implementation. Some scholars (e.g., Koch, 2007; Williams & Pollock, 2012) suggest studying ES and their surrounding ecosystems as "biographies" through longitudinal research on how and why the software features come to be as they are. Others argue for studying ES by examining the ecosystem of actors around it (Dittrich, 2014; Sarker et al., 2012). Most closely related to the second approach, my thesis shows that a way to organize the study of ES design in a manner geared toward understanding how to strengthen design is by seeing ES as a design infrastructure supporting implementation-level design.

The contribution is based on the research approach of the thesis. To examine how DHIS2 is currently designed and how this can be strengthened to better accommodate a diverse set of user organizations, I established and was responsible for sustaining what is referred to as a design lab at the outset of my thesis research project. The design lab team comprises myself, a group of researchers and master's students, and several DHIS2 practitioners. Our engagement started with active participation in a specific DHIS2 implementation project in India, where five master's students and I collaborated with the local partner. Later, we expanded our focus to examining the practices and challenges of several partners. Continuously, we have also actively engaged with the vendor, studying its practices, challenges, and strategic considerations. Furthermore, in collaboration with the vendor and the partners, we have explored concrete resources that may help address the challenges observed.

To explain the research approach of the thesis and by using the overarching theoretical framework developed to address RQ 1, I conceptualize the design lab as an approach to studying ES design in collaboration with practitioners. Concretely, I show how such an approach can involve diagnostic, design, and intervention-oriented research on processes of generic-level and implementation-level design and the resources of the design infrastructure. The approach can be relevant for studying ES design in other cases.

Second, my thesis presents an approach to engaged scholarship, which is relevant to the broader IS audience concerned with engaged scholarship in IS research (Mathiassen, 2017; Mathiassen & Nielsen, 2008; P. A. Nielsen & Persson, 2016). The design lab has involved several projects, including collaboration between various researchers and practitioners. The understanding of the real-world problem situation and the possibilities that lie in our collaboration with practitioners have evolved over time. Meanwhile, specific forms of engaged scholarship that are typically used to plan and carry out research projects in IS, such as action research (AR) and design science research (DSR), assume a rather detailed and static understanding of the problem of focus and the possibilities that lie in the collaboration with practitioners. Organizing the design lab using engaged scholarship as a methodological basis has allowed us to select different forms of inquiry (e.g., AR and DSR) as the understanding of the problem and the project has evolved. Based on this approach, my thesis offers a model for entering and organizing engaged IS research with relevance to a broader IS audience.

1.2 Papers

The thesis features four papers. The first three all play a part in addressing RQ 1, while the fourth relates to RQ 2. Table 1.1 summarizes the papers' titles, outlets, and roles in answering the RQs of the thesis.

Table 1.1. The papers and their roles in answering the research questions (RQs) of the thesis.

	Title, authors, outlet	**Role in answering the RQs**
1	Making Usable Generic Software. A Matter of Global or Local Design? Magnus Li and Petter Nielsen Scandinavian Conference on Information Systems (2019)	RQ1. Develops the early version of the theoretical framework to understand ES design. Identifies meta-design and implementation-level design as important elements of designing ES for a diverse set of user organizations. It shows how generic-level and implementation-level design work in tandem.
2	Generic Enterprise Software Implementation as Context for User-Oriented Design: Three Conditions and their Implications for Vendors Magnus Li Scandinavian Conference on Information Systems (2021)	RQ1. Examines implementation-level design as context for contextual design. Identifies challenges and conditions and discusses a set of implications for ES vendors.
3	Organizing Enterprise Software Ecosystems for Design: Considerations for Adopting a Platform Strategy Magnus Li and Petter Nielsen To be submitted to an international IS journal	RQ1. Examines how a platform strategy, that is, engaging the partners in designing, maintaining, and distributing generic add-on modules or apps that extend the features of the solution offered by the vendor, affects the dynamics of ES design. Identifies considerations for ES vendors adopting a platform strategy to support the ES design to accommodate a diverse set of user organizations.
4	Entering and Organizing Engaged Information Systems Research Magnus Li To be submitted to an international IS journal	RQ2. Establishes the methodological basis for the study. Defines a model for entering and organizing engaged IS research suited for conducting diagnostic, design and intervention-oriented research in collaboration with practitioners.

1.3 Thesis Structure

The rest of the thesis is structured in the following manner:

In **Chapter 2 – Related Literature,** I outline three perspectives offered by existing IS literature that are relevant for understanding ES design. The aim is to highlight key elements of the dominant perspectives of existing IS literature, which I later use to position my contributions in Chapter 6.

In **Chapter 3 – A Theoretical Framework for Understanding ES Design,** I elaborate on my understanding of the concepts of *design* and *contextual design* before I outline the overarching theoretical framework for the thesis, which aims to describe and explain

key aspects of ES design. The framework is developed based on an analysis of the empirical data and patterns in the related literature outlined in Chapter 2 and plays an important role in addressing the RQs of the thesis. In the chapter, I also use the framework to outline different modes of organizing ES design, which are relevant for understanding the findings related to challenges and considerations for adopting an ES platform strategy.

In **Chapter 4 – Research Approach**, I detail the research approach of the thesis while addressing RQ 2 on how researchers can study ES design in collaboration with relevant practitioners. I do so by first introducing engaged scholarship as the methodological basis of the thesis and the design lab before elaborating on the philosophical basis of my research project. I then present a history of how the design lab has evolved, explaining in detail how it is organized to study the design of DHIS2 in collaboration with the vendor and the partners. Based on this, I offer an explicit answer to RQ 2. Finally, I outline the dominant modes of data collection and analysis underlying the findings related to RQ 1 in more detail.

In **Chapter 5 – Findings,** I summarize the main findings related to RQ 1, which are structured into three sets: 1) overall dynamics of ES design, 2) conditions for contextual implementation-level design, and 3) challenges of and considerations for adopting an ES platform strategy. I summarize the chapter by offering an answer to RQ 1.

In **Chapter 6 – Contributions,** I discuss the three contributions related to RQ 1 to research concerned with ES design, and the contributions related to RQ 2, which involves both ES design and engaged scholarship in IS research more broadly. I also offer a set of practical implications of the findings for ES vendors, partners, and user organizations.

2. Related Literature

In this chapter, I offer an overview of the IS literature related to ES to help understand the phenomenon of ES design and later, in Chapter 6, to position my contributions. I begin by providing some background before I outline three perspectives of existing literature: ES implementation, ES generification, and ES ecosystems.

The literature related to ES design is challenging to navigate as relevant concepts and insights span multiple streams of IS literature, and several neighboring fields, such as organization studies (Wareham et al., 2014), software engineering (Dittrich, 2014; Sommerville, 2008), health informatics (Kaipio et al., 2017; Martin et al., 2007), and science and technology studies (Pollock et al., 2007). Moreover, much of the relevant literature examines specific types of generic ES, and, as argued by Light and Sawyer (2007, p. 528), this focus has "tended to override specific investigation into the more 'generic' aspects" of ES, and the body of knowledge makes up "a fragmented adhocracy" of conceptualizations and insights. Furthermore, relevant studies use different general labels, including ES (Elragal et al., 2020), packaged or product ES (Xu & Brinkkemper, 2007), generic software (Pollock et al., 2007), software suites (Ellingsen & Hertzum, 2019), and labels for various specific types of ES, such as ERP (Koch, 2007) and electronic health record systems (EHR) (Martin et al., 2007).

The challenge of designing generic ES can be seen in the light of the grander discussion in IS literature on *global versus local* (Rolland & Monteiro, 2002) and *standardization versus flexibility* (Ciborra, 2000; Hanseth et al., 1996; Monteiro et al., 2013). This literature explores the tensions of attaining global standards, while allowing flexibility for contextual design and the existence of idiosyncratic local use practices. In this stream of literature, standards are discussed broadly, and the need for standards may be motivated by several things, such as increased top-level management control and information integration. For ES design, as I shall show in this thesis, a central rationale for standardization is the idea of designing and maintaining software features that can be used by several user organizations.

In line with some researchers (e.g., Elragal et al., 2020; Seddon et al., 2010), I use the term ES to refer to the broad group of generic software solutions "designed to serve as a comprehensive solution to fulfill a broad range of essential organizational information processing needs on an organization-wide scope" (Haines, 2009, p. 183). To this end, ES tends to "incorporate modules that aid several organizational functional business areas such as planning, manufacturing, sales, marketing, distribution, accounting, financial management, human resources management, project management, inventory management, service and maintenance, transportation, and e-business operations and processes" (Elragal et al., 2020, p. 1). Thus, it refers to several types of organizational

software, such as ERP and customer relation management (CRM) solutions, used in domains ranging from manufacturing to health.

Rather than subscribing to a single stream of literature on ES, I have used insights across these perspectives to help make sense of the real-world problem situation explored in the thesis project. I have constructed the three perspectives based on a synthesis of a broad range of ES literature, with the aim of capturing the main ways in which ES design has been examined. The current chapter accordingly represents a synthesis somewhere between an organizing review and an assessing review (Leidner, 2018), examining a broad set of literature, with the aim of describing current knowledge, as well as identifying trends and gaps (the three perspectives, their key insights relevant to the RQ, and their limitations). In this work, I have also leaned on several literature reviews and papers that provide summaries of the pertinent perspectives (Bertram et al., 2012; Elragal et al., 2020; Halckenhäußer et al., 2020; Light & Sawyer, 2007; Singh & Pekkola, 2021; Tan et al., 2020; Williams & Pollock, 2012; Xu & Brinkkemper, 2007). The three perspectives are summarized in Table 2.1 and elaborated in the following sections. I end this chapter by discussing some of their limitations that are relevant for understanding the contributions of this thesis.

Table 2.1. Three perspectives in IS literature relevant for understanding ES design.

Perspective	**Main focus**
ES implementation	Highlights the challenges and efforts of implementing ES in specific user organizations and emphasizes the organizations' need and ability to change to align with the software
ES generification	Shows how an important part of ES design is the vendors' work of aligning and curbing diverse needs to offer generic software features and discourage individual customizations
ES ecosystems	Examines various actors involved in ES design and the relationships among them Suggests that vendors may encourage and support diverse needs by organizing a set of actors – an ecosystem – around the ES solution in order to engage in design and innovation

2.1 ES Implementation

The majority of IS studies on ES fall under the label "implementation studies" (Williams & Pollock, 2012), which generally examine "adoption-related issues from client perspectives" (Elragal et al., 2020, p. 3). This literature explores the challenges of implementing ES in organizations and develops guidelines, models, or critical success factors (Tan et al., 2020).

Many of the studies examine different stages of the implementation process, such as the procurement process from the perspective of user organizations, and offer guidelines and principles for selecting vendors (e.g., Damsgaard & Karlsbjerg, 2010). Others have developed frameworks that aim to help user organizations understand or guide the overall implementation process. Perhaps the most known and influential is Markus and Tanis's (2000, p. 184) "overarching framework, within which many specific questions can be asked and their answers integrated." This is based on "the perspective of an enterprise system-adopting organization […] to shed light on the questions facing the executive leadership of an organization considering whether, why, and how" to implement ES. Others offer life-cycle models extending into maintenance, how they evolve, and potentially retire (Elragal et al., 2020).

In general, these frameworks and guidelines portray a rather top-down implementation process, which dominantly involves changing organizational practices according to the software, thus contrasting contextual approaches to IT design. Accordingly, many studies emphasize organizational measures to help ease the transition to a new system and the new practices that must follow. For instance, some highlight the importance of support networks and other social arrangements (Sykes et al., 2014; Sykes & Venkatesh, 2017; Van Fenema et al., 2007).

From the entire breadth of the stream of implementation studies, three themes emerge that have particular pertinence for this thesis: misfits, customization, and ES implementation as a context of contextual design.

2.1.1 Misfits

Due to the generic nature of ES, misfits between generic software features and the needs and requirements of specific user organizations have received significant attention in the literature (Hustad et al., 2016; Soh et al., 2000; Strong & Volkoff, 2010; van Beijsterveld & Van Groenendaal, 2016). Early works primarily point out the challenge posed by these misfits, particularly when ES is designed for a global audience involving both organizational idiosyncrasies and more fundamental cultural differences (Soh et al., 2000).

Others have used various lenses to conceptualize what misfits are, and thus, when and why they emerge. For instance, Strong and Volkoff (2010, p. 733) conceptualize misfits as arising due to differences in the "latent structure" of the software that are not aligned with the structures of the organization, that is, rules and norms that are implicitly built into the software by the vendor based on assumptions or contexts of the ES origin. They further conceptualize two types of misfits: those that arise 1) when the relevant generic software features to sufficiently support the user organization are missing – a *deficiency* or 2) when the generic features enforce a less than ideal way of working – an *imposition*

(Strong & Volkoff, 2010). Along similar lines, Sia and Soh (2007) examine misfits using the lens of institutional theory.

Using these lenses, the literature offers various suggestions on how to deal with misfits. However, common to them all is their focus on how the user organization should, ideally, work to adapt their practices to the software. Meanwhile, only in the worst-case situations, for instance, when the software does not comply with national laws and regulations, should the software be adapted to the local context (Soh & Sia, 2008). Some studies also explore how organizations deal with misfits over time as the dust settles from the initial implementation (Berente et al., 2019; Berente & Yoo, 2012; Kharabe & Lyytinen, 2012). Furthermore, as portrayed in the literature, since ES implementation primarily involves setting up the software solution, there is less need for programming skills. However, this is replaced by the need for intimate knowledge about the generic software features and how these can be configured (Markus & Tanis, 2000). This gives rise to the possibility of perceived misfits due to a lack of knowledge about the ES, rather than actual misfits (van Beijsterveld & Van Groenendaal, 2016). Accordingly, some offer frameworks for assessing whether misfits are actual or perceived (van Beijsterveld & Van Groenendaal, 2016).

In any case, the general argument is that misfits represent an "inherent and inevitable aspect" of ES implementation (Sia & Soh, 2007, p. 569) and that misfits introduce challenges and adverse consequences for both the individual end users (in the form of usability issues) and the overall organization (Kallinikos, 2004a, 2004b; Soh & Sia, 2008; Topi et al., 2005; Van den Hooff & Hafkamp, 2017; Wong et al., 2016).

2.1.2 Customization

To address misfits, *customization* of the generic software features is often necessary during implementation. Several studies on misfits offer suggestions regarding situations when customization is appropriate (Hustad et al., 2016; Sia & Soh, 2007), that is, when the organization should "adjust its business processes to the [ES] and when [it] should adjust the [ES] to the business processes" (van Beijsterveld & Van Groenendaal, 2016, p. 369). A wide array of literature has examined the phenomenon of customization, primarily focusing on its different forms, the reasons for customization, and its negative effects. First, many identify a range of potential ways of customizing ES with different pros and cons (e.g., Brehm et al., 2001; Hustad et al., 2016). The three most dominant are configuration, modification of the source code, and extension by building add-ons using user exits or application programming interfaces (APIs) (Haines, 2009).

Configuration involves setting parameters in the software that are predefined by the vendor, including the definition of organizational hierarchies, data reporting format and contents, and module selection (Sommerville, 2008). It is thus limited to the "'switching on and off' of functionality that is part of the blueprint of the software" (Light, 2001, p.

417). Although configuration seldom requires any programming and thus incurs low costs, it often offers very limited flexibility to adapt the software. Extension involves building custom features on top of the generic solution. Aided by an API, it allows developing custom functionality and UIs. Modification of the source code refers to changes that are not supported by the vendor. Although it may afford full flexibility to change the software, this is often a complex endeavor, and as I shall show, it entails both immediate and long-term costs for the user organization.

Although ES in principle offers indefinite flexibility for customization by allowing modification of the source code, utilizing this potential is often highly discouraged (Light, 2001; Pollock & Cornford, 2002; Sestoft & Vaucouleur, 2008). This is due to the challenges it introduces in terms of keeping up to date with the vendor's upgrades to the generic software features. As articulated by Haines (2009, p. 182), "a key reason for choosing to implement an ES is the hope that overall IT costs will be reduced by shifting a large part of the software development and maintenance burden to the ES vendor." Modifying the source code will effectively impede this ability. Therefore, the advice is to limit software adaptation to the standardized configuration facilities in which vendors often offer support when upgrading the software to new versions (Khoo et al., 2011; Sestoft & Vaucouleur, 2008).

In addition to the cost challenge, a key argument in the literature is that by customizing, the organization may fail to benefit from "bring[ing] in some of the best practices of the industry" (Parthasarathy & Sharma, 2016, p. 19), supposedly manifested in the generic software features (Koch, 2007). Unnecessary customizations may thus be triggered if the implementation team fails to stand up against end users' resistance to change (Rothenberger & Srite, 2009; Singh & Pekkola, 2021)

2.1.3 ES implementation as context of contextual design

Although the literature on misfits and customization acknowledges that adaptations of generic software features are often necessary, it presents adaptation as an evil necessity to ideally be avoided because it is associated with major immediate and long-term costs related to development and maintenance work. ES implementation is thus presented as an infertile context for contextual design, which necessitates modifications to IT. In general, and despite several calls for research, ES implementation has received little attention as a context for software design and development when compared with traditional bespoke projects (Baxter & Sommerville, 2011; Bertram et al., 2012; Dittrich, 2014; Light & Sawyer, 2007; Sommerville, 2008).

However, a few studies have examined the use of contextual approaches to design and development during ES implementation (Hocko, 2011; Magnusson et al., 2010; Pries-Heje & Dittrich, 2009). For instance, Magnusson et al. (2010) and Pries-Heje and Dittrich (2009) report about projects where participatory design was used to organize

the implementation project. Others, such as Hocko (2011) and Vilpola (2008), present user-centered design as a potential method of organizing ES implementation. However, some studies do not acknowledge the defining issue of costly customization and thus assume that design and innovation mainly relate to changes in organizational routines and practices. Those that do acknowledge it point to few remedies. Pries-Heje and Dittrich conclude that "ERP systems present a serious challenge for the design process, as they already provide a relatively comprehensive body of functionality that constrains the design space" (2009, p. 52). In line with this, several scholars argue that ES implementation represents a difficult context for contextual design due to the limited technical design flexibility, constraining the design process. For instance, Martin et al. (2007, p. 55) report that "when straightforward technical solutions to usability problems cannot be found, they are inevitably turned into training issues" and thus addressed by adapting organizational practice rather than the software. Others argue that any contextual and user-oriented design that aims to affect the software is primarily relevant in the procurement and vendor selection process. As pointed out by Ellingsen and Hertzum, "after the vendor has been selected the preparations for implementing the system are strongly shaped by the product already available from the vendor, including its configuration possibilities" (2019, p. 7).

Although the literature offers important insights on how to organize ES implementation to consider and involve end users in the process, it presents a very limited possibility for changing the technology according to specific needs. If so, inevitably, these processes would primarily be about changing the organization according to the software rather than the other way around (Kallinikos, 2004a; Martin et al., 2007).

2.2 ES Generification

The second perspective on ES in IS literature is born out of a criticism of the one-sided focus on implementation from the perspective of the adopting organization (Koch, 2007; Williams & Pollock, 2012). The literature stream offers accounts of the ES vendors' design of generic software features, which broadly involves identifying similarities among strategically important user organizations while curbing those needs that are unique or only relevant to a few (Kallinikos, 2009).

Central in these discussions is a seminal study by Pollock et al. (2007), which shows how a prominent ES vendor employs a set of strategies, referred to as *generification* work. In essence, this involves the attempt to identify shared needs among a strategic selection of user organizations, which are "generified" and turned into generic software features that are "ready to travel" between contexts (Kallinikos, 2009, p. 915). Meanwhile, needs unique to one or a few organizations are avoided. To make this acceptable by user organizations, generification work involves several social strategies for curbing unique needs. For instance, in the case examined by Pollock et al. (2007), a

key form of requirement elicitation is referred to as an *alignment workshop*, where user organizations are gathered to discuss their needs. An aim of gathering the user organizations in common discussions is to provoke a "witnessing effect," where the user organizations can realize the level of generic relevance or uniqueness of their requirements and thus accept that these would not be addressed in the generic solution. As Pollock et al. (2007, p. 269) argue, generification as such is largely based on curbing diversity:

> Encouraging users to carry out organizational change to align with the system is an important strategy for managing the user base, and also a way to reduce the need for the further accumulation of particular functionality. It is a method, in other words, of moving users towards the 'organizationally generic'.

A large set of user organizations and the combined effort to define the generic solution while curbing diversity give rise to a wide range of challenges and interesting social dynamics around ES design, explored and conceptualized in the literature. For instance, the vendor must find ways of prioritizing which user organizations it should accommodate the most and will thus have the greatest influence on the design of generic software features. Such organizations are often selected based on their economic or strategic importance (Gizaw et al., 2017; Nicholson et al., 2019), for instance, if accommodating the organization could help the vendor enter new industry segments (Pollock et al., 2007). In addition to maintaining strategic relationships with user organizations to inform the generic design and to curb diversity, vendors work strategically to include them in activities related to sales and diffusion. For instance, Pollock and Hyysalo (2014) show how user organizations adopting a certain ES solution perceive themselves as on the same boat as the vendor. Thus, they are willing to contribute not only by informing the design of the generic software but also by acting as "referent organizations" that help attract new adopters. User organizations do so with the hope of yielding influence over the design process to secure the sustainability of the generic solution over time by participating in expanding the user base.

When faced with too many diverse and incompatible needs, the vendor may segment the market, offering different generic modules to accommodate various types of use cases and organizations. For instance, when bringing modules initially developed for the US healthcare market to hospitals in the UK (Mozaffar et al., 2018), each segment and the corresponding module are subject to their own generification processes, specific to a country such as the UK.

As some ES solutions serve thousands of user organizations, only a few can hope to influence the generic design. Most organizations are left out and must deal with the software as it is (Koch, 2007). To help in the "mass curbing" of the needs of the larger audience that does not participate in the more intimate alignment work, vendors often

brand their solutions as embodying "best practices" for how to structure organizational work (Koch, 2007; Swan et al., 1999; van Groenendaal & van der Hoeven, 2008; Wagner et al., 2006). The argument is that the user organizations are best served by adapting to the software "so that less efficient organizations can use it to raise the standard of their internal business processes" (Singh & Pekkola, 2021, p. 6744). However, the idea of ES representing the "best practice" has been heavily criticized and shown to be often based on a matter of coincidence related to the organizations used as the basis for the design, rather than representing a universal "best" form of organizing (Wagner et al., 2006). As argued by van Beijsterveld and Van Groenendaal (2016, p. 370), "there is no single best way to design organizational structures"; "best" is rather contingent on a company's internal and external situations. Promoting ES solutions as embodying best practices can therefore be viewed as much a strategy of curbing diversity as a genuine belief in offering the best way to arrange organizational work processes:

> Embedding the full set of specific work procedures from a larger range of organizations would prove ineffective. ERP companies like SAP have here been successful in creating a belief that their product represented 'best practice', thus creating a situation in which local users were driven on the defensive, since specific details of the setting were construed as unnecessary barriers to development. (Koch, 2007, p. 432)

2.3 ES Ecosystems

The third and final perspective suggests that vendors may also encourage and support diversity. The first two streams mainly acknowledge two types of actors: the user organization, which is highlighted in the first perspective, and the vendor, which is the primary focus in the second. The ecosystem perspective focuses on how some ES vendors not only interact directly with a set of user organizations but "open up" (Farhoomand, 2007) and seek to establish and maintain an ecosystem of actors around their solutions (Antero & Bjørn-Andersen, 2013; Dittrich, 2014; Sarker et al., 2012). Central to such an effort is the involvement of *partner* organizations, which may be engaged in two capacities that are key to ES design: 1) as value-added resellers (VARs) or "ES consultancies" that specialize in supporting user organizations in implementation (Jæger et al., 2020) and 2) as independent software vendors (ISVs) that design, maintain, and distribute generic add-on modules or apps that extend the features of the solution offered by the primary vendor.

For the first capacity as VARs, the vendor seeks to involve partners that can accommodate user organizations in implementing the ES. The vendor benefits by avoiding the work of an intimate relationship with every user organization, and the ES can thus potentially reach a larger audience. Furthermore, partners can often cultivate

"far greater expertise in their native markets" (Wareham et al., 2014, p. 1196). Combined with their expertise in the given ES, they may also offer domain-specific and industry-specific expertise beyond what is possible for a single vendor (Sarker et al., 2012). The partners may themselves be large software houses, such as Accenture (Staub et al., 2021), and can therefore also complement the implementation services with additional expertise related to software development and digitalization. For the vendor, engaging partners as VARs means that they must invest efforts in "meta-design" (Dittrich, 2014), that is, in designing resources that aid the partners in further design during implementation. A central part of this is the technical flexibility to configure and extend the software (such as the *customization* capabilities outlined in Section 2.1). Particularly, vendors often invest in resources that aid in extensibility to allow partners to design and develop custom modules on top of the solution to meet specific user needs (Antero & Bjørn-Andersen, 2013; Dittrich, 2014; Roland et al., 2017). Another key element is the provision of knowledge resources that support the partners in utilizing the generic software features and their adaptation capabilities (Foerderer et al., 2019; Sarker et al., 2012).

In addition to cultivating an ecosystem of partners specializing in ES implementation, a vendor may adopt a platform strategy akin to that of consumer software platforms, such as iOS and Android. The key aim of adopting such a strategy is to engage the partners as ISVs. In doing so, the vendor hopes to extend the portfolio of generic software features beyond what it could do on its own (Magnusson & Nilsson, 2013; Wareham et al., 2014), in turn improving the ability to accommodate a diverse set of user organizations. For instance, both SAP and Salesforce have adopted platform strategies (Farhoomand, 2007; Kauschinger et al., 2021), and Salesforce is particularly known for having successfully established a vibrant ecosystem of ISVs around its ES solution, offering more than 3,000 apps (Staub et al., 2021). If a vendor is successful in establishing such an innovation platform, as in the case of Salesforce, the ES ecosystem is transformed into a multi-sided market of buying and selling enterprise apps. With this, the vendor hopes to benefit from cross-sided network effects, where the more ISVs and thus generic apps are available for the ES platform, the more attractive it is for adoption and vice versa.

Adopting a platform strategy is not a new situation; as reported, ES vendors as early as 2000 pursued a strategy of "componentization," which involved organizing core data model features as forming a "backbone to which the offerings of other vendors can be connected" (Markus & Tanis, 2000, p. 179). However, with significant attention to consumer software platforms, IS research has more recently developed an interest in understanding the dynamics of such ecosystems (de Reuver et al., 2018), including those surrounding ES solutions (Staub et al., 2021; Wareham et al., 2014).

In addition to the meta-design for supporting the partners in extending the generic software features with apps, the literature highlights two important ingredients of an ES platform strategy. First, the vendor often offers standardized resources for supporting and regulating the distribution of the apps designed and maintained by the partners, frequently in the form of an *app marketplace* (Magnusson & Nilsson, 2013). Second, while outsourcing design and maintenance work to partners, the platform owner must invest effort in "the careful governance of complementors in order to profit from their development outcomes" (Foerderer et al., 2019, p. 120). Adopting a platform strategy must therefore involve developing and sustaining a set of governance mechanisms, which must seek a balance between securing partners' capacity to design novel capabilities and ensuring control to avoid any potential deterioration of the quality of the ES (Huber et al., 2017; Wareham et al., 2014). Such governance is often exercised through constraints in the development resources, by regulating the app marketplace (Huber et al., 2017), and by enforcing rules and regulations, such as through partnership programs (Wareham et al., 2014). For a more detailed overview of the literature on ES platform strategies, see Paper 3.

2.4 Limitations of the Existing Perspectives

I end this chapter by summarizing and discussing some limitations of the three perspectives. Each perspective offers valuable insights into the practice and challenges of ES design. The first perspective highlights the challenges of misfits and the efforts to address them during implementation in specific user organizations. A limitation of the first perspective is its exclusive focus on implementation, mainly from the viewpoint of the adopting user organization. Meanwhile, as argued by Williams and Pollock (2012, p. 5), "many issues regarding the material character of [ES] artifacts are determined outside the setting of technology adoption." The perspective is thus "inadvertently producing only 'partial' understandings of these systems" (Williams & Pollock, 2012, p. 1). Furthermore, the perspective primarily emphasizes the organizations' need and ability to change according to the software while presenting a misfit as an "inherent and inevitable aspect" (Sia & Soh, 2007, p. 569). Meanwhile, little attention is paid to technology design (Berente et al., 2019). Those studies that focus on customization mainly present it as an evil necessity that should be avoided if possible. The underlying sentiment is thus how to best avoid rather than support contextual design and agility for user organizations on the level of implementation.

The second perspective offers valuable insights into how an important part of ES design by the vendor entails aligning and curbing diversity and offering "generified" features. It also highlights how generic design does not represent a "design from nowhere" (Suchman, 1993, p. 29), without a basis on "real" user needs, but is based on encoding generic requirements built on an intricate relationship between the vendor and several

user organizations. Whereas the implementation studies are criticized for being limited to a one-sided perspective favoring implementations, the generification stream suffers from almost exclusively focusing on the vendor side. In doing so, it portrays design as exclusive to the vendor and draws a sharp line between software *design* as a vendor-led activity and *implementation*, which, as in the first stream, seems to mainly involve organizational change according to the software. Consequently, neither is able to fully capture the dynamics between the two contexts of design, and the stream does not account for how contextual design involving shaping and innovation of software can unfold during ES implementation.

Related to both the first and the second perspectives, Mozaffar et al. (2018, p. 91) notes:

> While there is increasing attention to the misalignment between technology and organization, much of this research is limited to emphasizing consequences of either 'global' design and development, or 'local' user customization practices. We argue that a dual user-vendor perspective is needed to understand the bridging between the 'global' act of vendors (generification) and the 'local' operations of adopters (e.g., workarounds).

The third perspective suggests that vendors may also encourage and support diversity by cultivating an ecosystem of partners engaging in design. A few researchers (e.g., Dittrich, 2014; Koch, 2007) highlight the relevance of an ecosystem perspective for studying ES design. However, the majority of the literature on ES ecosystems pays little explicit attention to the problem of designing to accommodate the needs of a diverse set of user organizations. Rather, it is occupied with examining other particular aspects of ecosystem dynamics, such as challenges associated with app marketplaces (Magnusson & Nilsson, 2013) and different modes of governance (Huber et al., 2017). Furthermore, most of the literature is mainly on a mission to generalize to technology ecosystems (including consumer platform ecosystems, such as Android and iOS). The reason for examining ES is often to serve as a case of an "extreme form of technology ecosystem" (Wareham et al., 2014, p. 1196), highlighting the traits that are general and shared, regardless of the type of platform ecosystem, while the specific traits related to ES design are downplayed.

I now turn to outlining the theoretical framework of the thesis; based on insights from existing literature, it aims to help understand ES design.

3. A Theoretical Framework for Understanding ES Design

I now outline a theoretical framework for understanding ES design. The purpose of the framework is to help analyze how ES is designed to serve the specific needs of a diverse set of user organizations by conceptualizing important processes and resources. In existing literature, both understanding and studying how ES is designed are argued as challenging since ES and the relevant surrounding resources involved in its design and implementation represent "extremely complex sociotechnical assemblages encompassing a huge variety of elements that are shaped over space and time" (Williams & Pollock, 2012, p. 14). Consequently, useful conceptualizations must help analyze "how they [(ES solutions)] are inserted into organizational practices and also how they are evolving over time and across multiple sites of suppliers, users, and specialist intermediaries" (Williams & Pollock, 2012, p. 2). A key part of this thesis project has been to develop a theoretical framework that aids in studying and understanding the design of DHIS2, which I claim is relevant to understanding important processes and resources as part of ES design in general. The framework is elaborated and used as the basis for the analysis in Papers 1–3. While Paper 1 offers the initial version of the framework, Paper 3 provides the most recent and extensive one.

The framework has been developed through abductive cycles of analyzing empirical observations and examining similarities and patterns in existing literature. I elaborate more on the data collection and analysis process behind the framework in Chapter 4. Before I turn to the framework, I offer definitions of *design* and *contextual design* as these concepts are central to the thesis. After presenting the framework, I use it to briefly explain different modes of organizing ES design based on existing literature, which are relevant for understanding what is meant by a *platform strategy*. Later, in Chapter 5, where I present the findings that address RQ 1, I empirically substantiate the framework by illustrating the different concepts at play in the DHIS2 ecosystem.

3.1 The Concepts of Design and Contextual Design

Design is a central concept in this thesis. I employ a broad definition of design to include all activities that intend to devise "courses of action aimed at changing existing situations into preferred ones" (Simon, 1996, p. 130). Accordingly, in the ES context, design is an integral part of the development and maintenance of generic software features, the work of configuring and extending these features during implementation, and the intentional change of the user organization according to the features of the generic software. Thus, in line with (among others) Dittrich (2014), I perceive design

as something unfolding on several levels and in multiple constituencies of the ecosystem surrounding ES.

This broad definition of design applies to many kinds of activities. Related to IS research, design is a central part of IT projects involving developing, modifying, and extending a piece of technology. Design is also an important part of organizational change programs that involve strategic changes to organizational practice, with the aim of improving organizational efficiency (Markus, 2004). Both are relevant during ES design, which (as shown in Chapter 2) often involves shaping both technology and organizations.

By *contextual design*, I refer to a specific form of design, embodying a certain set of principles that could be found in many user-oriented and agile approaches to software design and development, such as user-centered design (Gulliksen et al., 2003; Norman, 2013), activity-centered design (A. Williams, 2009), work-oriented-design (Blomberg et al., 1996), design thinking (Dorst, 2011), and socio-technical design (Baxter & Sommerville, 2011). Although these approaches differ in several ways regarding their means, ends, and design scope, they share an emphasis on designing technology based on the practices and needs of a concrete set of end users. The rationale is to ensure the relevance and usability of IT, and it involves developing requirements based on inquiries into the context of (future) use, rapid and evolutionary prototyping, and frequently evaluating the prototypes with end users to define both the function and the form of IT artifacts.

3.2 Two Forms of ES Design

I now turn to the theoretical framework. A pattern in existing literature, which corresponds to the case of DHIS2 (as shown later), is the involvement of two types of design processes in shaping the ES that eventually faces end users in a given user organization: generic-level and implementation-level design.

First, ***generic-level design*** refers to processes of designing "generic software features such as data models, functionality, and user interfaces, intended to be used across multiple user organizations" (Paper 3, p. 10). The key aim of generic-level design is to benefit from economies of scale by sharing the same software features among several user organizations. Accordingly, it necessarily involves some degree of standardizing software features that are "ready to travel" (Kallinikos, 2009, p. 915) between contexts. As seen from the ES generification perspective of existing literature (Gizaw et al., 2017; Pollock et al., 2007), design often involves identifying shared traits and common ways to support the needs of multiple user organizations. An aspect of generic-level design is thus *generification*, that is, the attempt to align the needs of user organizations to standardize and mass produce and maintain software.

Generic-level design also involves meta-design (Dittrich, 2014), that is, design with the intent to support further shaping of the generic software features during implementation (e.g., configuration and extension facilities). The output of generic-level design is thus both generic software features, such as functionality and UIs (intended to be used more or less "as is" by end users after implementation), and adaptation capabilities to support configuration and extension during implementation.

Second, ***implementation-level design*** refers to processes of "designing software according to the needs of a particular user organization based on [the] generic software features" offered by the processes of generic-level design (Paper 3, p. 10).

With the term *implementation-level design*, I do not intend to cast all activities during ES implementation as design, but I aim to capture the activities whose purpose is to devise "courses of action aimed at changing existing situations into preferred ones" (Simon, 1996, p. 130) on the level of implementation. During ES implementation, this involves improving some aspects of the user organization with the implementation of technology by configuring and extending generic software features and by designing those organizational aspects to align with the software.

I also stress the distinction between implementation-level design and what some studies explore under the label *design in use* and end-user *appropriation*, which involves the work that end users engage in to make technology function during *use time* (Draxler & Stevens, 2011; Fischer & Giaccardi, 2006). Whereas these activities are important in making the systems work and could be considered as constituting a form of design (Pipek & Wulf, 2009), they do not comprise the end users' primary activity. Rather, they constitute a form of articulation work that is necessary to leverage the technology as a tool to achieve other ends. In contrast, implementation-level design is carried out by professional software designers, developers, and IT project managers by leveraging generic software features, with the aim of undertaking software design. Akin to traditional bespoke software development projects, it involves activities such as requirement gathering, prototyping, software configuration and development, and roll-out (Sommerville, 2008). What makes it different from bespoke projects is its aim to primary rely on the generic software features offered by generic-level design.

3.3 ES as Design Infrastructure for Implementation-Level Design

The third concept of the theoretical framework is ***design infrastructure,*** which refers to "the set of generic software features and knowledge resources that support [implementation-level design] in constructing a solution for the individual user organization" (Paper 3, p. 10). The design infrastructure thus links the processes of generic-level and implementation-level design. Because existing and new generic software features undergo constant design and maintenance by the processes of generic-

level design, the design infrastructure continuously evolves. The knowledge resources could include documentation, educational materials, certification programs for partners, support arrangements, and anything that offers support to the implementation-level design process (Xu & Brinkkemper, 2007). Figure 3.1 illustrates the relations among the three concepts.

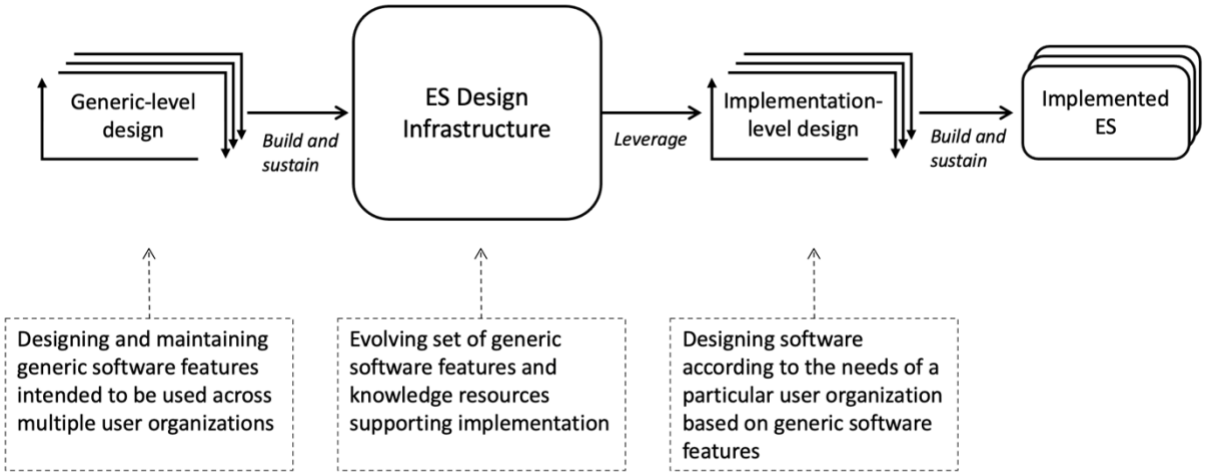

Figure 3.1. A theoretical framework for understanding ES design.

I base my understanding of the term *infrastructure* on existing conceptualizations in IS literature (Hanseth & Lyytinen, 2010; Pipek & Wulf, 2009; Star & Ruhleder, 1996). Infrastructures are defined as socio-technical, open, and shared collections of resources that serve a supporting function for some activity that spans multiple actors. They "shape and are shaped by conventions of practice" (Pipek & Wulf, 2009, p. 454) and are evolving, recursive, and layered. In the context of ES design, with the term *design infrastructure*, I highlight the infrastructures' supporting role in implementation-level design. I thus argue that the notion is effective in capturing both the role (*design*) and the nature (*infrastructure*) of the generic ES, offering the basis for implementation-level design.

Perceiving sets of software features as infrastructures supporting design has also been suggested in the prior literature on end-user development and design-in-use appropriation, for instance, under the label *appropriation infrastructure* (Stevens et al., 2009). These appropriation infrastructures support the end users in appropriating software solutions as part of the inevitable articulation work that comes with using technology. In contrast, design infrastructures, as defined in this thesis, offer support to professional software design and development processes during ES implementation.

3.4 Modes of Organizing ES Design

The two types of design processes (generic-level and implementation-level design) seem to comprise a consistent pattern of ES design, as portrayed in the related literature. However, the literature on ES ecosystems (e.g., Dittrich, 2014; Sarker et al., 2012; Wareham et al., 2014) shows different modes of organizing ES design, related to how design and maintenance work are carried out by one or partitioned among several actors. I briefly highlight these modes, with the help of the theoretical framework since the distinctions are important to understand the findings related to the challenges of and considerations for adopting a platform strategy.

First, some ES solutions are not designed within a partner ecosystem; thus, both generic-level and implementation-level design are exclusively controlled by a single vendor, collaborating with user organizations through dyadic or arms-length relations. An example of an ES that follows this mode is the widely used EHR system Epic, where the vendor designs and maintains the totality of the generic software features for the "core solution," while engaging in implementation-level design with the specific user organizations (Ellingsen & Hertzum, 2019). This is also dominantly how ES design is portrayed in the first and the second perspectives of the related IS literature. Figure 3.2 illustrates this mode of organizing ES design.

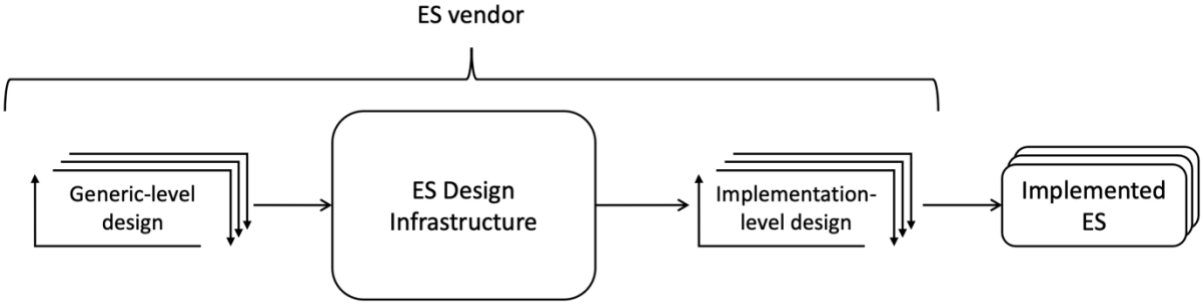

Figure 3.2. ES design organized with a single vendor carrying out both generic-level and implementation-level design (no ecosystem).

Second, the ES vendor may cultivate an ecosystem comprising partners that specialize in implementation-level design (as VARs). In this mode, the ES vendor is the only actor engaging in generic-level design, but, in contrast to the first mode, the partners that may be geographically or domain-wise closer to various user organizations are the main enactors of implementation-level design. This mode, as illustrated in Figure 3.3, is how the DHIS2 design is presented in Papers 1 and 2 of this thesis.

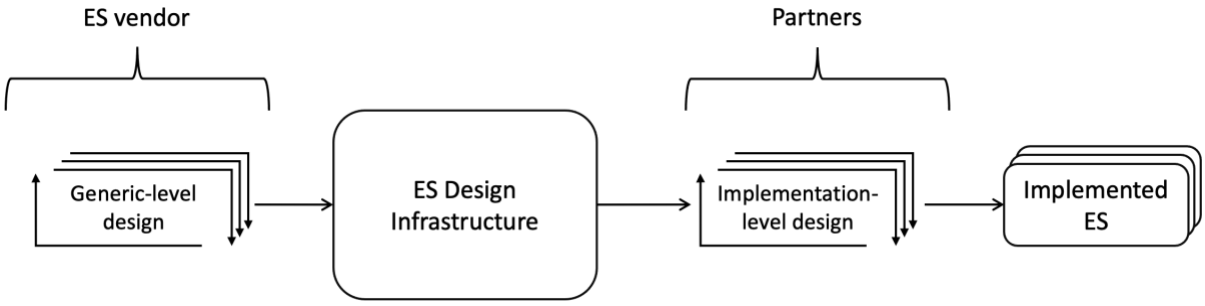

Figure 3.3. ES design organized with a single vendor carrying out generic-level design, while partners conduct implementation-level design (ecosystem).

Third, the ES vendor may adopt a platform strategy to open up the conduct of the generic-level design to partners (as ISVs). In this mode, both the vendor (now often referred to as the "platform owner") and the partners may design and maintain the generic software features that form part of the design infrastructure (Figure 3.4). In contrast to the second mode, in the third, the ES resembles what Tiwana (2013) refers to as a *real* platform in terms of being multi-sided. In other words, partners may offer generic software features to the larger audience of partners and user organizations engaging in implementation-level design, while the latter audience benefits from these contributions. Accordingly, in this thesis, when I discuss adopting a platform strategy, a key aim is for partners to be actively engaged in generic-level design, not only in the leverage of vendor-developed features during implementation-level design. It is this mode of organizing design that I examine in Paper 3 of the thesis, where I study the adoption of a platform strategy.

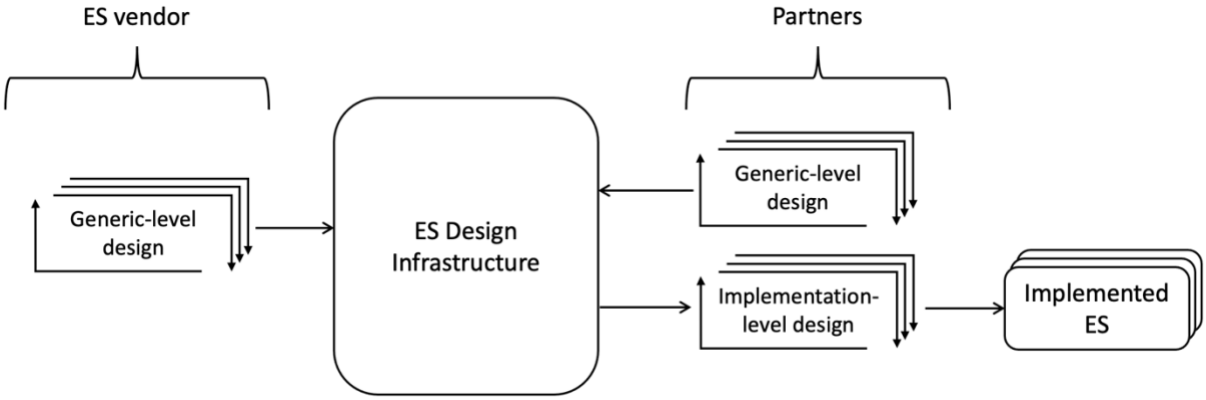

Figure 3.4. ES design organized with both the vendor (now the "platform owner") and the partners carrying out generic-level design, while the partners conduct implementation-level design (platform ecosystem).

An important takeaway from the above outline of different modes of organizing ES design is that *platform* is but one way of organizing ES design (Bertram et al., 2012; Mousavidin & Silva, 2017). There are multiple alternatives for partitioning design and maintenance work among the actors (e.g., a single vendor in charge of both generic-level and implementation-level design, an ecosystem including partners specializing in implementation, and platform ecosystems supporting partners in offering generic apps). In my thesis, and in line with several researchers (e.g., Tiwana, 2013), I only consider ES an innovation platform when partners or other third parties are included in offering generic software features that form part of the design infrastructure. Thus, a platform strategy in essence entails opening up to third parties in carrying out generic-level design. In contrast, if only a single vendor offers generic software features and the partners or other third parties only engage in implementation-level design, the characteristics of innovation platforms, such as *multi-sidedness* and *cross-sided network effects* (Tiwana, 2013), are not pertinent. I shall return to this point in the DHIS2 context in Section 5.3

Having established an understanding of different modes of organizing ES design, I now outline the research approach of the thesis.

4. Research Approach

In this chapter, I outline the research approach of the thesis project. The chapter has a twofold purpose. First, I offer an account of the research project to show how the data have been collected and analyzed to answer RQ 1. Second, while doing so, I summarize the findings related to RQ 2 on how researchers can study ES design in collaboration with relevant practitioners. Accordingly, I attempt to offer a broad account of the evolution and nature of the DHIS2 Design Lab. Meanwhile I also delve into the details of the primary modes of data collection underlying the findings related to RQ 1. To do so, I first briefly introduce DHIS2 as my empirical case. Second, I introduce engaged scholarship as the methodological basis for the thesis and present pragmatism as the philosophical basis. Third, I describe how I have established and sustained the DHIS2 Design Lab as the engaged research *engine* for exploring the DHIS2 design, which I summarize with an explicit answer to RQ 2. Finally, I offer some details on the primary data sources and the analysis approach to addressing RQ 1.

4.1 Case: DHIS2

While I shall explore the design of DHIS2 in detail in Chapter 5, here, I provide a brief background of DHIS2 and the central actors of focus in the analysis.

4.1.1 The DHIS2 core solution

Typically, DHIS2 is implemented to support the collection, analysis, and presentation of routine and aggregate health information. Increasingly, it is also implemented to support case-based data collection, such as registering information about patients, commodities, or lab samples. To this end, the generic DHIS2 "core solution" used as the basis of all implementations comprises a generic data model, functionality, and UIs that support data collection and presentation. The core solution is divided into several modules or apps with a shared underlying data model. An API also allows developing custom apps during implementation. As part of the core solution, the apps support commonly relevant activities, such as routine or case-based data entry (Figures 4.1 and 4.2), creating information dashboards (Figure 4.3), maps (Figure 4.4), reports, and analytical tools to create graphs and pivot tables. Furthermore, a set of apps support configuring the core solution, such as managing users and access, defining what data to be reported and by whom, and organizational hierarchies. These are used for configuring the solution during implementation-level design.

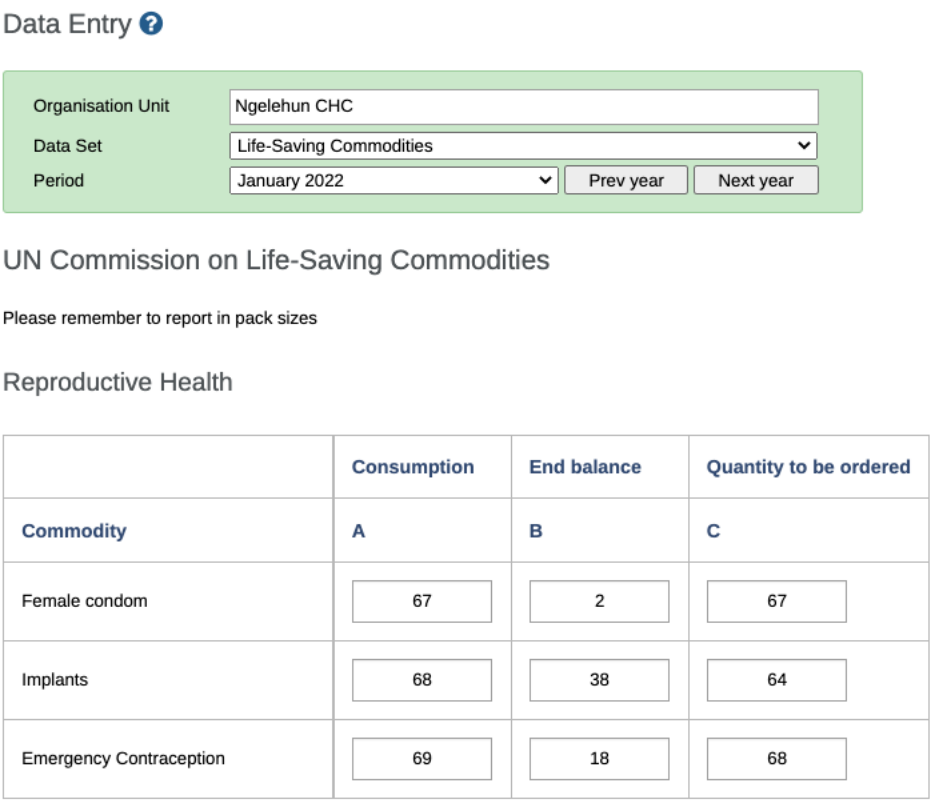

Figure 4.1. An example of a routine data entry screen in DHIS2 (not showing real data).

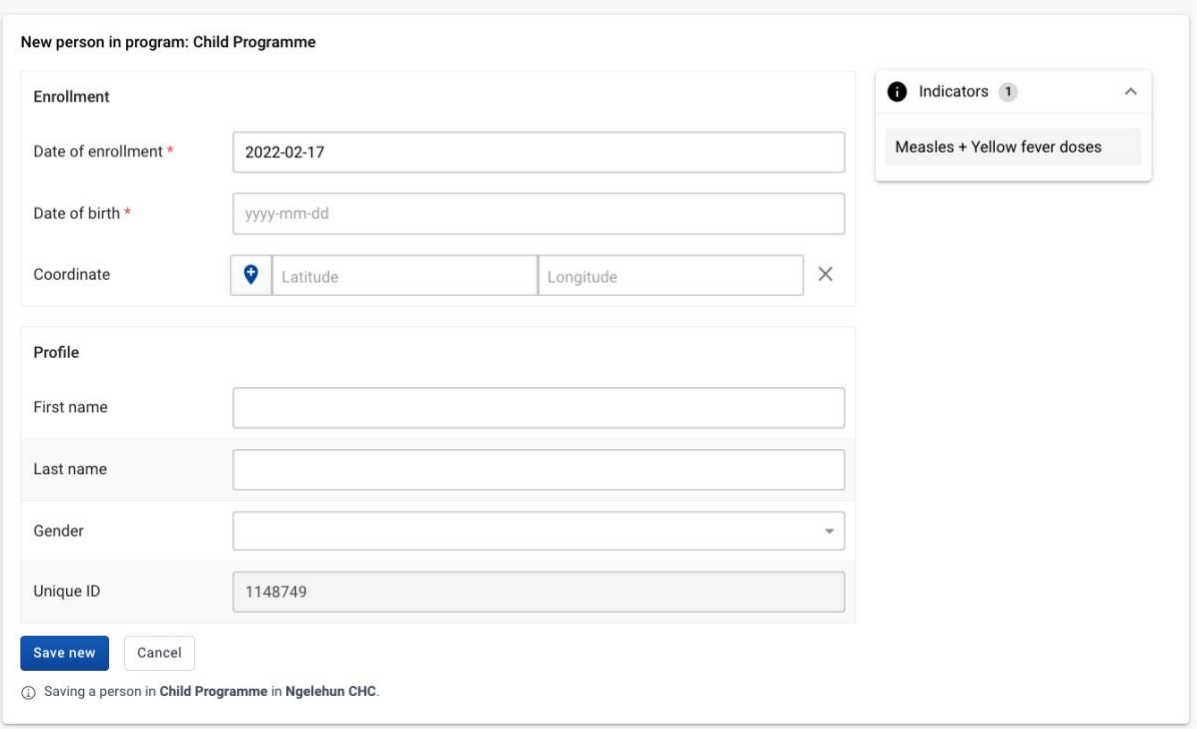

Figure 4.2. An example of a case-based data entry screen in DHIS2.

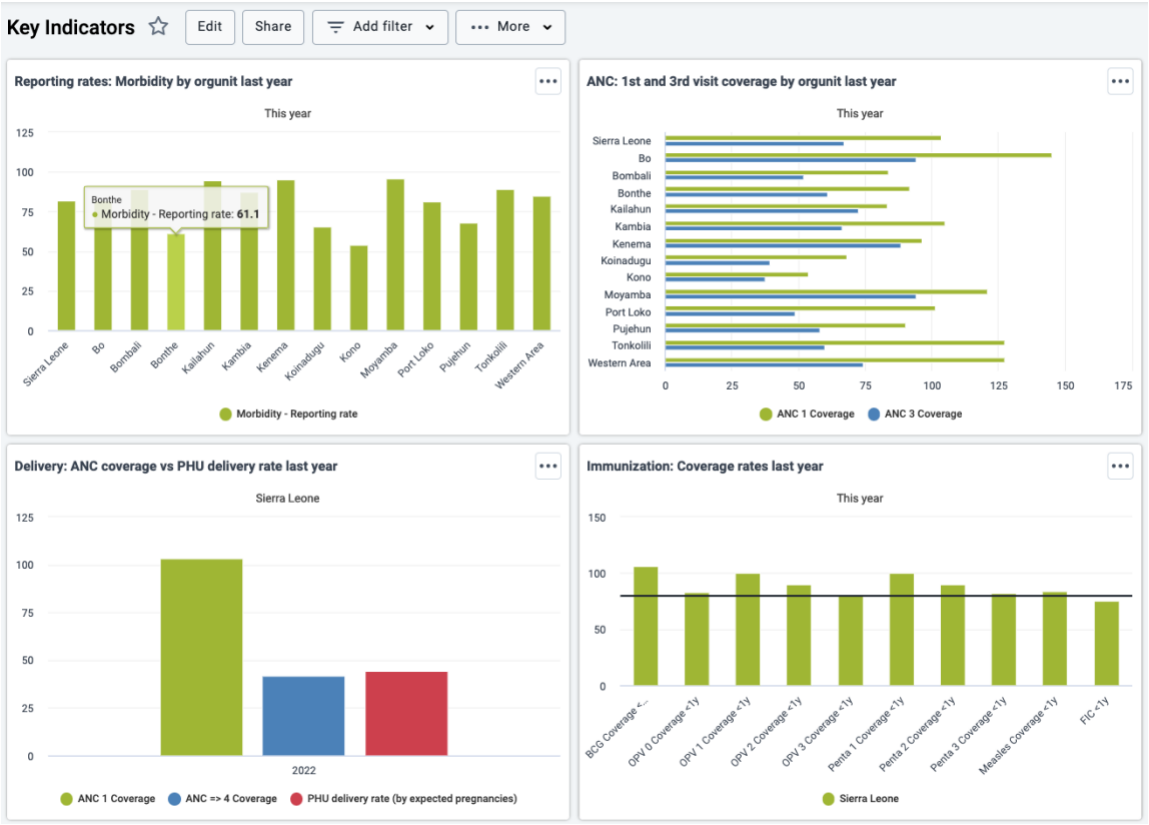

Figure 4.3. An example of a dashboard screen in DHIS2 (not showing real data).

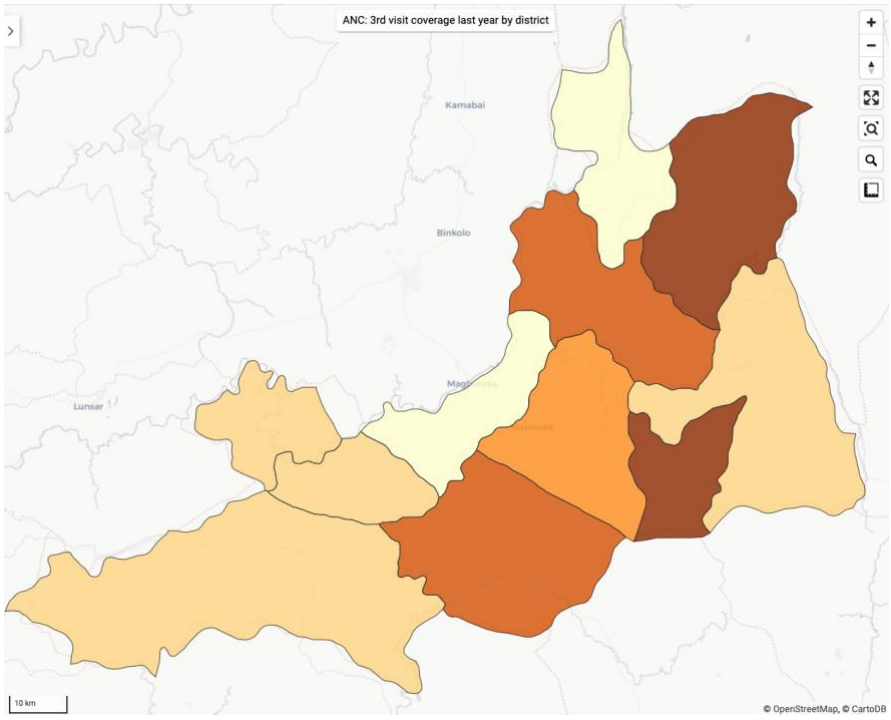

Figure 4.4. An example of a map in DHIS2 showing antenatal care (ANC) coverage in a region (not showing real data).

DHIS2 has somewhat of a special history compared with many other ES solutions. It is offered as an open-source solution, involving no licensing fees for user organizations, and is developed through a research program called the Health Information Systems Programme (HISP), which has been operating for more than two decades. HISP started in post-apartheid South Africa in the early 1990s as a collaboration among researchers from the University of Oslo (UiO), local researchers, and stakeholders from the South Africa's Ministry of Health (MoH). The aim was to develop an integrated health management solution that could help address challenges of fragmentation in South Africa's existing health management information system. The software that is now known as DHIS2 was a central part of these efforts, and proven successful in South Africa, the software later "traveled" to several neighboring countries and is now implemented in public and private organizations across more than 80 countries in Africa, Asia, South America, and Europe. Accordingly, the HISP program is today a global research network. Figure 4.5 shows a few examples of implementations of DHIS2, illustrating the diverse contexts and use cases that it serves.

The UiO still plays a central role in the HISP program and hosts the team of designers, developers, and project managers in charge of offering the generic DHIS2 core solution. Furthermore, the IS research group at the UiO conducts research related to the design, development, and implementation of DHIS2 in a variety of countries, and the design lab and I are part of this research group. Thus, the team at the UiO includes both researchers and practitioners, and I refer to the group of practitioners as the *core team* and the vendor of DHIS2.

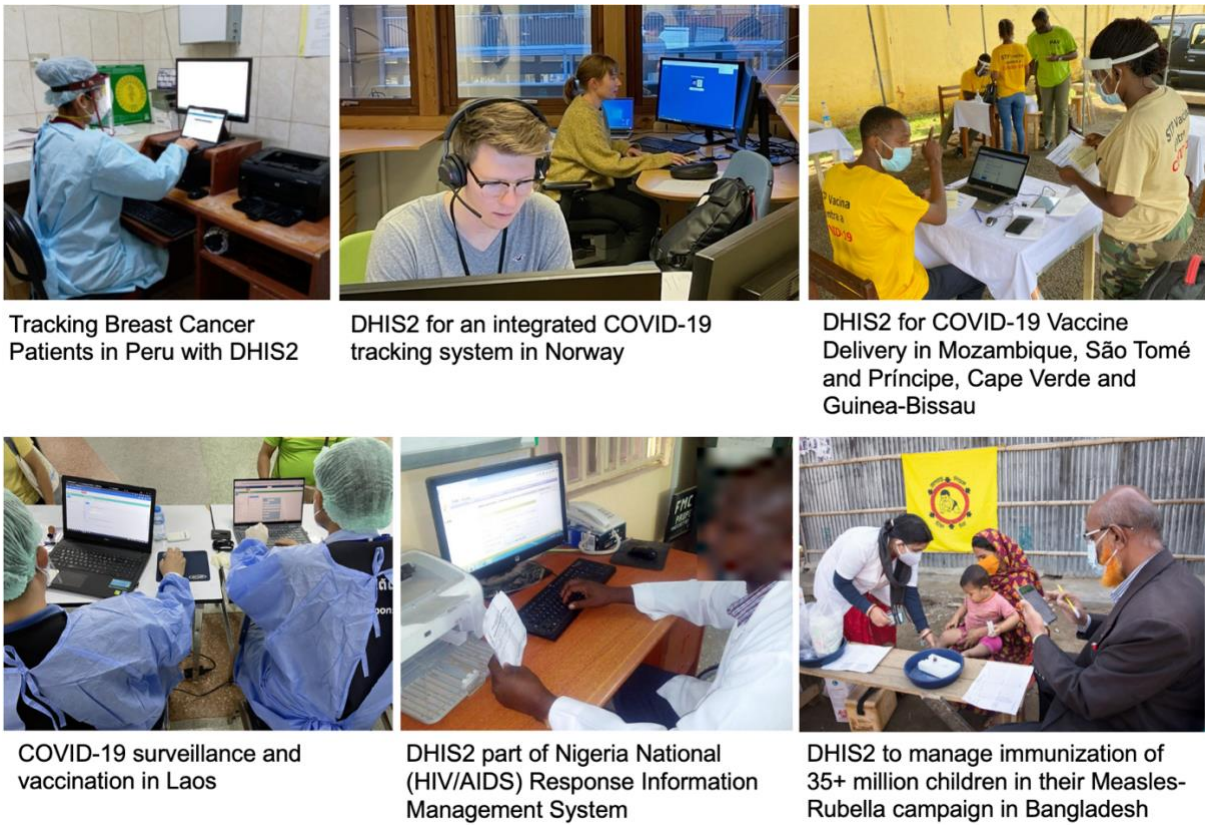

Figure 4.5. Examples of DHIS2 implementations in diverse contexts and use cases.

4.1.2 Central actors

In this thesis, I focus on three central types of actors related to DHIS2: the core team as the vendor, the partners, and the user organizations. The core team comprises several groups of software professionals at the UiO. First, *product managers* manage teams who design, develop, and maintain specific parts or modules of DHIS2, such as Analytics, Front-end, Back-end, and Tracker (case-based data collection module). Each of these teams includes several *developers*. Working accross all teams is a lead user experience (*UX) designer* who is in charge of the UI design and manages a design system that includes the style guides and UI components (e.g., buttons, forms, tables) used across the teams for consistent UI design. Other members of the core team are in charge of project and implementation support, including *implementation experts* who support partners and user organizations in certain implementation projects.

The second type of actors comprises the partners that specialize in implementing DHIS2 for user organizations. Although independent firms, the partners have strong ties with the core team in Norway and participate in defining requirements and in strategic discussions on the future direction of the DHIS2 core solution (a matter that is explored in more detail in Chapter 5). The partners are located in various countries; for this thesis, I have primarily engaged with seven (based in India, Tanzania, Malawi, Uganda,

Mozambique, the US, and Sri Lanka). Typically, the partners employ *implementers*, who may work as both project managers communicating with the user organizations and designers gathering requirements, designing and evaluating prototypes, and configuring the core solution according to the users' needs. As the dominant portion of their projects are related to health, many implementers have backgrounds in clinical or public health, in addition to their competence in DHIS2 and in organizing implementation projects. Furthermore, the partners employ *developers* who develop custom software features if needed during implementation. The partners typically employ staff specializing in server maintenance, security, and management as well.

Each partner is involved in a multitude of projects. A central user organization for many partners is the MoH in the country or region of each partner. The partners often have long-term contracts with these user organizations for the design, maintenance, and support of health programs, such as HIV management, tuberculosis monitoring and treatment, antenatal care, and disease surveillance (e.g., of COVID-19). In this case, the DHIS2 solution may be implemented as one shared instance cutting across multiple health programs or having separate instances for each. Additionally, the partners continuously compete for contracts from other public and private organizations in the health sector and beyond. For instance, the partner in Tanzania serves the Tanzanian MoH in several health programs, as well as user organizations in agriculture, traffic policing, and water and sanitation. In India, the partner serves the MoH in several Indian states MoHs in neighboring countries, such as Bhutan and Nepal. The partner in India is also engaged in projects in African countries, such as Egypt and Kenya. Meanwhile, the partner in Mozambique serves the national MoH, and beyond Mozambique, the partner concentrate on the Lusophone countries of Africa, such as Cape Verde, Guinea-Bissau, and Angola (e.g., as seen in the top-right picture of Figure 4.5).

The common denominator of all user organizations is that they have needs involving data to be reported, analyzed, and presented, where the generic data model, functionality, and UIs of DHIS2 can work more or less out of the box or serve as valuable starting points to be customized and extended. Table 4.1 summarizes the three central types of actors focused on in this thesis.

Table 4.1. Three key DHIS2 actors in this thesis.

Name	Role
Vendor (core team)	Designs, develops, and maintains the DHIS2 core solution Offers knowledge resources, such as documentation, implementation guidelines, and question/answer (Q/A) forum to support implementation
Partners	Specialize in implementing DHIS2 for user organizations Plan and carry out implementation projects involving activities such as requirement gathering, prototyping, configuring the core solution, and potentially, developing custom software features
User organizations	Use DHIS2 to support collection, analysis, and presentation of data and information, primarily related to the health domain, as well as other domains, such as agriculture, logistics, and education

For more detailed information about DHIS2 and examples from various use cases, see https://dhis2.org/. A demonstration of DHIS2 is available at https://play.dhis2.org/, where anyone can test an actual working version of the system. For a more detailed overview of the history of DHIS2 and the HISP program, see Adu-Gyamfi et al. (2019) and Braa et al. (2004).

4.1.3 DHIS2 as a case of ES

As an ES, DHIS2 and the design processes and the actors that surround it shares the general challenge of designing and maintaining ES features that can be used among several user organizations. With this, it shares many of the issues and strategic considerations with widely known and commercially licensed ES, such as SAP and Salesforce, including dividing the design between generic and implementation levels, managing a partner network, and attempting to adopt a platform strategy. These general traits and challenges, combined with the unique access to study and collaborate with both the vendor and the partners of a globally used ES, make it a fruitful case for studying ES design more generally.

DHIS2 also has particular traits as an ES. The core team is not a commercial vendor but funded through research and development funds and is a part of a long-term research program. The core team is also based in a university and thus, tightly integrated with researchers. The researchers at the UiO have strong ties with the partners and the core team and actively collaborate with the practitioners on matters related to research and practice. Meanwhile, the implementation partners mainly engage in implementation projects in the *global south*, which may be funded by the user organizations directly or with the support of international development agencies as donors. As with all cases,

some elements are therefore not immediately generalizable from DHIS2 to other ES, particularly elements tied to its funding and its close relation to a research institution. I return to some of these elements in the Limitations section of Chapter 6 (Section 6.6.)

4.2 Engaged Scholarship

This thesis is based on engaged scholarship (Mathiassen, 2017; Van de Ven, 2007). A key feature of engaged scholarship is that problems are defined and examined on the basis of real-world problem situations and in collaboration with practitioners. As such, it is "a participative form of research for obtaining the different perspectives of key stakeholders (researchers, users, clients, sponsors, and practitioners) in studying complex problems" (Van de Ven, 2007, p. 9). This is contrasted to research initiated and driven by a gap in academic literature, followed by a strategic case selection that best allows studying the assumingly relevant question. Engaged scholarship accordingly emphasizes problem formulation as an empirical and continuous activity throughout the research process to ensure that researchers explore problems that are relevant to practice. It is suggested as particularly relevant to IS as an applied research discipline, whose primary objective should be to develop knowledge of relevance to IS practice, as argued by many scholars (e.g., Conboy et al., 2012; Mathiassen, 2017).

Engaged scholarship could be carried out in different forms, for instance, with the aims of exploring practitioners' practices and challenges, designing artifacts, such as policies, or introducing and evaluating organizational change. For the field of IS research, Mathiassen and Nielsen (2008) distinguish among three relevant forms of engaged scholarship that correspond to a set of methods widely used in IS: 1) what they refer to as practice studies, which I here equate with case studies (CSs), 2) AR, and 3) DSR. Furthermore, in Paper 4, I argue the fourth distinct form to be action design research (ADR) (Sein et al., 2011). The different forms are useful for developing distinct types of knowledge to respond to the real-world problem situations and the related knowledge interest of researchers and practitioners. Additionally, the forms differ in their requirements regarding researcher–practitioner collaboration, ranging from engaging participants in interviews and discussions to an intense collaboration in introducing and evaluating organizational change. Table 4.2 summarizes situations where the four forms are relevant and feasible (from Paper 4).

Table 4.2. Four forms of engaged scholarship and when they are relevant and feasible (from Paper 4).

	Relevance – what is produced?	**Feasibility – what is required?**
CS	Descriptive and explanatory knowledge of relevance to the real-world problem situation and a specific concern in academic literature	Study phenomenon in a real-world problem situation as an attached *insider* (participant in the system under study) or as a detached *outsider* (e.g., interviews, questionnaires, focus groups)
DSR	Artifact(s) relevant to the real-world problem situation, and prescriptive knowledge from the process and/or the result of designing and evaluating the artifact(s) of relevance to a specific concern in academic literature	Design and evaluate artifact(s) (e.g., models, methods, tools) to address a problem in the real-world problem situation
AR	A change in practice relevant to the real-world problem situation, and descriptive, explanatory, and/or prescriptive knowledge from the process and/or the result of an organizational change of relevance to a specific concern in academic literature	Plan, introduce, and evaluate organizational change to address a problem in the real-world problem situation
ADR	Artifact and change in practice from artifact introduction relevant to the real-world problem situation, and prescriptive knowledge from the process and/or the result of designing and evaluating the artifact(s) of relevance to a particular concern in academic literature	Design, introduce, and evaluate artifact(s) that reflect(s) "the influence of users and ongoing use in context" (Sein et al., 2012, p. 40) to address a problem in the real-world problem situation

Often, engaged research projects in IS are planned using a specific form of engaged scholarship, such as AR or DSR. However, a problem with selecting a particular form of engaged scholarship at the outset of a research project (which I elaborate on in detail in Paper 4) is that it requires an understanding of the real-world problem situation and the possibilities that lie in collaboration with practitioners. Nonetheless, IS research is often not a matter of solving a static problem but about developing a better understanding of the problem, which helps researchers ask new questions, in turn pointing to potential solutions (Nielsen & Persson, 2016; Van de Ven, 2007). Such an evolving understanding of the problem situation and the potentials for collaboration with practitioners have been characteristic of the research project of this thesis. I started with an open problem formulation of how to improve the usability of DHIS2 in implementations and an agreement with one partner organization in India to participate in a specific implementation project in order to explore this challenge in practice. Later, based on insights from this first set of inquiries and through abductive cycles of

identifying patterns and similarities between the empirical findings and the related academic literature, the problem has been reformulated and elaborated in several iterations. The research process has in itself been one of developing a deeper appreciation of the problem, while continuously negotiating with relevant practitioners on possibilities for further inquiry. I have accordingly used engaged scholarship as a methodological basis for the research project and have seen the various forms of what I refer to as *engaged inquiry* (i.e., CS, AR, DSR, and ADR) as tools to be employed based on the evolving appreciation of the real-world problem situation and the emerging possibilities to collaborate with practitioners. This overall model is presented in Figure 4.6 and is explained in greater detail in Paper 4.

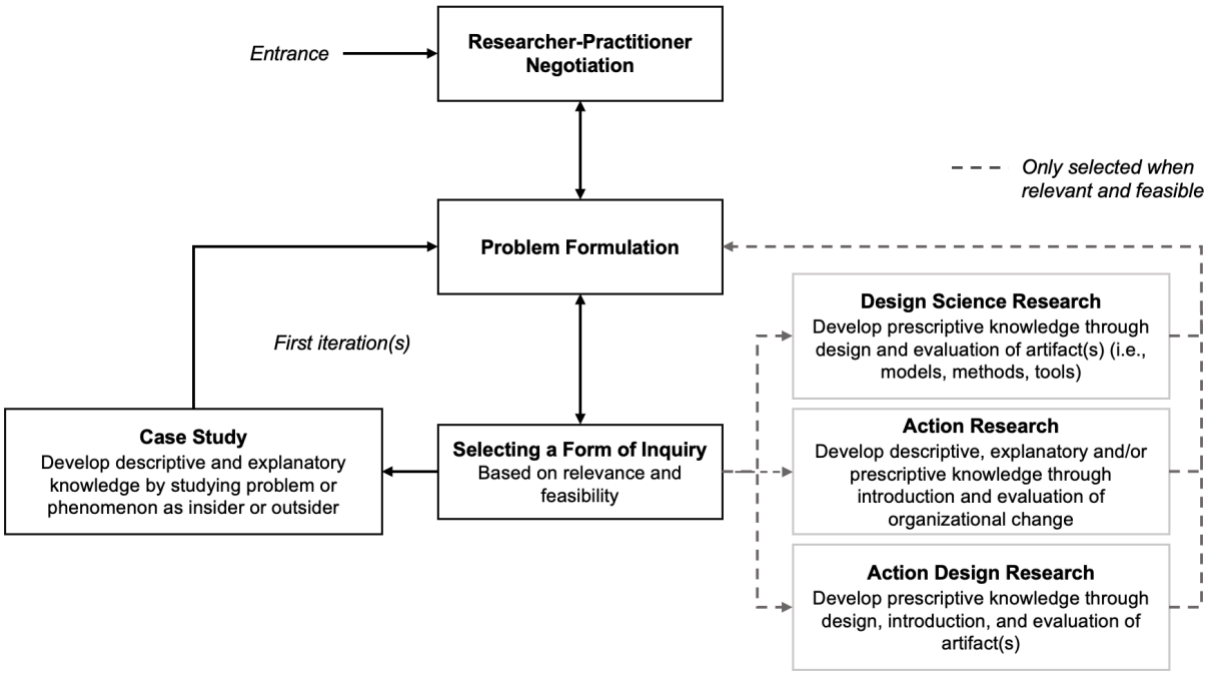

Figure 4.6. The model used for organizing the engaged scholarship research project (from Paper 4).

Three important activities of the research process that have been frequently revisited are researcher–practitioner negotiation, problem formulation, and, based on these, selection of a form of inquiry. Furthermore, in the model, an insider or an outsider CS is always used as the starting point when initiating the investigation of a new (sub-)problem. As outlined in Section 4.4, the research project has included several researchers and master's students collaborating with DHIS2 practitioners. Accordingly, the overall research project has utilized several forms of inquiry in sequence and in parallel, all exploring particular aspects of designing DHIS2, while feeding into the

overall problem formulation. The specific set of findings presented in this thesis is primary based on insider and outsider CSs, which I will return to in Section 4.5.

4.3 Pragmatism as Philosophical Basis

Before turning to the details of the research project, I offer some reflections on the philosophical basis for the research, which I position as pragmatism. Since pragmatism is less known to many IS scholars than paradigms, such as interpretivism, positivism, and critical realism (Goldkuhl, 2012), I invest a few paragraphs in explaining some of its relevant tenets.

More than a hundred years earlier than the publication of Van de Ven's (2007) book on engaged scholarship, pragmatism arose from a similar kind of criticism of contemporary philosophy as that of engaged scholarship of social and organizational research. In observing the age-old debates about the nature of reality and the nature of knowledge, a group of scholars representing different fields and interests looked with skepticism at the advances made in answering the questions and at the potential value of the debates (Bacon, 2012). Pragmatism grew out of the idea that philosophers should stop concerning themselves with questions that bear no practical consequences and differences in practice. The central figures of the tradition argue that questions are only relevant if they make a difference in practice. Accordingly, philosophy and science should seek "to gain the kind of understanding which is necessary to deal with problems as they arise" rather than to try to uncover the "antecedently real" (Dewey, 1969, p. 14, as cited in Bacon, 2012, p. 47).

With its emphasis on practical consequences and on the development of useful knowledge, I argue that pragmatism is a particularly relevant philosophical basis for engaged scholarship research. This is in line with others' views (e.g., Van de Ven, 2007), and pragmatism has witnessed growing interest in the social sciences (Morgan, 2014) and organizational science (Lorino, 2018), and although somewhat implicit, it has been influential in IS research (Ågerfalk, 2010; Goldkuhl, 2012), particularly as the basis for AR and DSR (Goldkuhl, 2020).

The tradition of pragmatism is diverse and filled with debates and disagreements. The main figures include Charles Sanders Peirce, John Dewey, and Williams James, but later figures, such as Willard Quine, Richard Rorty, and Susan Haack, are influential voices of the tradition (Bacon, 2012). All of these proponents have been in significant disagreement with one another over many of the central tenets of pragmatism, making it difficult to present a single coherent version. Based on discussions in social, organizational, and IS research (e.g., Goldkuhl, 2012) and leaning on Bacon's (2012) summary of the key ideas and debates on pragmatism, I briefly outline a set of pragmatist arguments and ideas. These are related to the nature of knowledge and the

scientific enterprise that I argue is particularly relevant to engaged IS research, which underlies my research in this thesis.

4.3.1 Four relevant traits of pragmatism

First, similar to interpretivism, pragmatism argues that we, as humans and researchers, are not bystanders looking *at* the world but agents operating *in* it. This means that we can never assume a neutral spectator's view of the world. As articulated by Quine (1980, p. 79), "we can improve our conceptual scheme, our philosophy, bit by bit, while continuing to depend on it for support; but we cannot detach ourselves from it and compare it objectively with an unconceptualized reality." Whereas interpretivists highlight the study of humans' subjective understanding of phenomena, pragmatists stress that the meaning of an idea or a concept comes down to its consequences for human action. Thus, "a society must be seen and grasped in terms of the action that comprises it" (Goldkuhl, 2012, p. 136). Pragmatism thus posits that knowledge is deeply embedded in and intertwined with human experience, action, and practice. It theorizes that "knowing and doing is indivisibly part of the same process" and that the scientific enterprise should accordingly be "concerned with action and change and the interplay between knowledge and action" (Goldkuhl, 2012, p. 136).

Rather than seeking the "antecedently real," pragmatists thus put human action and *experience* at the center of both the development and the use of knowledge (Bacon, 2012; Lorino, 2018). However, pragmatism does not deny the existence of a material reality or the possibility that both social and physical phenomena may exist without our individual or collective knowledge of them. There is a material world that shapes and is shaped by us. As a species, we have evolved and developed bodily functions, material tools, knowledge, and social practices (including norms, values, and language) to cope with our existence in the world. Both the material and the social aspects of the world, where we operate as part of it, offer corrective sanctions and rewards, based on our actions (Bacon, 2012). Some scholars thus position pragmatism between *empiricism*, holding that an objective world "commands thought," and *idealism*, holding that "subjective thoughts construct the world" (Bechara & Van de Ven, 2007, p. 56).

Second, as knowledge is perceived as playing a practical role in helping us cope with our environment, theories and concepts are regarded as tools to help us better understand, explain, predict, and improve real-world problem situations (Bacon, 2012; Goldkuhl, 2012). Thus, "the function of intelligence is therefore not that of copying the objects of the environment, but rather of taking account of the way in which more effective and more profitable relations with these objects may be established in the future" (Dewey, 1931, as cited in Goldkuhl, 2012, p. 140). This means that the value of a theory or conceptual framework does not lie in how accurately it corresponds or mirror some objective "untainted" reality. Rather, "[a]s in the case of all tools, their value

resides not in themselves but in their capacity to work shown in the consequences of their use" (Dewey, 1969, p. 163, as cited in Bacon, 2012, p. 49)

To offer an example that is close to the topic of this thesis, the value of concepts such as "design infrastructure" or "platform" does not lie in how well they correspond to a mind-independent reality. Rather, their value is tied to supporting researchers and practitioners in dealing with certain real-world problem situations to "make one a more sensitive observer of details of action, better at asking useful questions, more capable of seeing the ways actions are patterned, and more adept at forming systemic hypotheses and entertaining alternatives" (Cronen, 2001, p. 30).

Third, well-justified knowledge is argued by figures such as Peirce to be "intimately connected to the means by which beliefs are arrived at" (Bacon, 2012, p. 152). Moreover, the scientific enterprise should seek what Haack (2000) refers to as *genuine* inquiry, which requires knowledge to answer to experience, which in turn involves answering to *reality* (Bacon, 2012). Some knowledge is thus regarded as superior to others in being better aligned with experience and as a result, being more practically relevant. For example, if we hold the belief that humans could fly, when repeatedly put to the test, experience would most probably tell us otherwise. Similarly, the belief that the same ES, without any adaptation, could work perfectly across a diverse set of user organizations fails the test of experience, as reported by hundreds of implementation studies.

Disregarding the ideal of knowledge as corresponding to the antecedently real, in order to arrive at the best possible knowledge, Dewey argues that researchers should seek *warranted assertibility*, which is still, as articulated by Bacon, (2012, p. 55) "a matter of arriving at well-grounded beliefs which answer to our objective situation rather than our individual needs." Several pragmatists highlight the methods of natural sciences as supreme at arriving at the best possible knowledge. This is not due to their specific methods of attempting to uncover untainted *truth* but to their principles of *seeking* to be a disinterested truth-maker, which involves being directed to wherever the evidence leads and being ready to abandon beliefs in the face of evidence (Bacon, 2012).

Fourth, an important implication of pragmatists' views of knowledge and action is that these open the door to methodological pluralism. Methods, possibly including both qualitative and quantitative types, should be selected based on their potential merit in developing useful knowledge of the phenomena in question (Goldkuhl, 2012; Morgan, 2014). Whereas interpretivism is appropriate for examining questions related to how people make sense of IS, and positivism is suitable for studying questions related to prediction (Vidgen & Braa, 1997), pragmatism may serve as a philosophy underlying both kinds of questions as long as the inquiry may lead to useful knowledge. However, as knowledge ultimately has to answer to experience, the methods that allow studying

phenomena based on the experiences of participants or directly subjecting oneself as a researcher to experience seem particularly appropriate. It could thus be argued that insider CS, AR, and ADR represent ideal approaches to developing and evaluating knowledge that is subjected to the test of answering to experience. However, these would inevitably be tied to a specific context. A benefit of the more detached-outsider forms is that a researcher could better examine experience across longer timeframes and instances (Goldkuhl, 2012). Another benefit of being an outsider in the particular practice of the study is that of having fewer "habits" tied to the phenomenon, and it is thus more likely that taken-for-granted beliefs are subject to inquiry.

4.3.2 Why pragmatism?

A natural and very pragmatic question to ask is what difference a pragmatic basis makes to the research practice of this thesis compared with more well-established paradigms, such as logical positivism and interpretivism. First, pragmatism turns its focus to the development of descriptive, explanatory, and prescriptive knowledge that aims at being constructive in coping with real-world problem situations by engaging with real-world problem situations.

Such knowledge is best developed by focusing on action, experience, and practice. The data collection and analysis in this thesis are therefore characterized by studying action and experience both directly (as an insider) and indirectly (through the experiences of participants). When doing so, pragmatism allows balancing between empiricism and idealism and thus examining mechanisms and events that could be claimed to exist, regardless of our intersubjective or individual awareness of them without claiming a spectator's view. In contrast to interpretivism, it can thus help justify going beyond "understanding the subjective meanings of persons" (Goldkuhl, 2012, p. 137). Whereas interpretivism primarily seeks knowledge through interpretations of our "compatriots'" interpretations of the world (Walsham, 2006, p. 320), positivists seek knowledge that presents mirroring of reality. In contrast, pragmatism highlights knowledge as comprising tools and instruments for coping with our existence in the world. Accordingly, the aim of using and developing theory and conceptual schemes in this thesis is to best make sense of the real-world problem situation. The thesis aims to add to the buffet of potentially useful perspectives and does not claim that their merit lies in offering the best possible correspondence to an objective reality but in their utility to make sense of and cope with ES design as a real-world problem situation. In Chapter 6, I return to the pragmatist merit of my contributions.

4.4 The DHIS2 Design Lab

I now turn to the details of the research project and begin by elaborating on the DHIS2 Design Lab. The account of the DHIS2 Design Lab is important in explaining the research process of the thesis and in answering RQ 2.

The research project of the thesis and the design lab started with discussions with my supervisor on how I could best explore a phenomenon interesting to me and relevant to DHIS2. I had already studied design practices related to DHIS2 implementations in my master's research project. Based on the experiences from the master's project, we agreed that understanding how the usability and relevance of DHIS2 could be strengthened in implementations represented an interesting and relevant focus. We further agreed that establishing a design lab, including researchers and master's students who would collaborate with DHIS2 practitioners, could make an interesting *engine* to explore the design of DHIS2.

The *DHIS2 Design Lab* has subsequently been central to the research project. While I elaborate on the primary data collection methods used to obtain the findings presented in this thesis in the next section, in this section, I provide an overview of the DHIS2 Design Lab and how it has served as the engaged research engine of the project. Whereas the findings presented in this thesis mainly stem from a subset of the inquiries carried out in the design lab, the findings of other inquiries conducted by myself, other researchers, and master's students at the UiO, in collaboration with practitioners, also serve as an important broader basis for the findings. The master's students in our research group at the UiO are planning and undertaking a research project spanning 1.5 years, which is intended to result in a thesis offering a contribution to a pertinent concern in academic literature. Within HISP, the students often carry out their empirical work by studying aspects related to DHIS2, and from its initiation, the master's students have been central to the work of the design lab.

At the initiation of this research project, how the design lab could best be organized for studying and potentially helping improve the design of DHIS2 was an open question. As mentioned in Chapters 2 and 3, ES has been studied from multiple perspectives, and it has been argued that it is difficult to study ES design because it is subject to "shaping […] distributed in time and space" (Williams & Pollock, 2012, p. 3). Meanwhile, several forms of design laboratories (or labs) are defined and discussed in academic literature (Alavi et al., 2020; Binder et al., 2011). Some studies focus on exploring design *products*, such as usability labs (Bødker & Buur, 2002;Nielsen, 1994) or labs for citizen-centric service or health technology innovation (Bergvall-Kareborn & Stahlbrost, 2009; Sahay et al., 2018). Others focus on exploring design *processes*, that is, how to strengthen the processes of designing better products (Binder & Brandt, 2008). In any case, the term *lab* is typically used to refer to the experimental nature of

the approach to exploring novel ways of designing, and a common trait is that labs are organized as collaborative environments where various stakeholders are involved in exploring products or processes. However, academic literature offers little guidance on how such labs could be organized for studying ES design. Thus, in addition to exploring the DHIS2 design, an important part of the research process of this thesis has been to identify how the design lab could best play a role in studying and strengthening the DHIS2 design, and the evolution of the design lab is tightly linked with the development of the theoretical framework and the findings of the thesis.

4.4.1 A brief history of the design lab

The design lab as a project started with the open question of how to strengthen the usability and relevance of DHIS2 to end users in implementations, along with an opportunity to collaborate with a partner in India by participating in a specific implementation project. The project had been ongoing for three years, and DHIS2 had already been implemented and used across some 3,000 health facilities in a large Indian state. The partner was in the process of fine-tuning the implemented system by adding new functionality to respond to emerging needs and to address a set of usability issues highlighted by the user organization. I became involved, together with a group of five master's students, with the aim of participating in addressing the usability issues. We began by participating in several meetings and visiting health facilities to identify and define a concrete set of challenges that we could help tackle. The initial idea was to frame the project as AR and to examine how we could strengthen the partner's design practices by experimenting with different methods of usability design. However, it quickly turned out that the challenges in addressing the usability issues were as much related to the limited design flexibility of DHIS2 and how the implementation project was organized in terms of scope and structure as it was due to the lack of usability focus in the partner's design practice. With limited relevance and feasibility of introducing and evaluating the use of usability methods, we decided to reframe the project as an insider CS and focused on examining the broader challenges and obstacles to conducting usability-oriented and contextual design during the implementation of DHIS2.

The engagement in India helped develop the first version of the overarching theoretical framework of the thesis and resulted in the publication of Paper 1 of this thesis. It also resulted in several master's theses that examined various challenges in organizing implementation projects and in attempting to introduce contextual design methods to software organizations. The experiences in India and the theoretical framework also helped in making sense of how the design lab could best play a role in strengthening the DHIS2 design, indicating that strengthening it must consider both generic-level and implementation-level design and how the two types of processes work in tandem. The

conceptualization thus helped raise new questions related to the nature of both processes and the design infrastructure linking them.

In parallel to the engagement in India, examining the potential for contextual implementation-level design of other partners emerged as relevant. Together with a new group of four master's students, I thus visited partners in Tanzania, Malawi, and Mozambique, focusing on questions such as the following: What characterizes implementation-level design? What does the process typically look like, and what activities does it include? What practices and challenges of DHIS2 partners are related to contextual implementation-level design?

Continuing the visits to partners, a master's student visited Malawi for a longer duration to explore methods for contextual design of IT for rural health workers. The project was initially planned as AR, but due to the COVID-19 pandemic and the subsequent travel restrictions, the project was turned into an outsider CS inquiry. Here the student involved several implementers from partners and representatives from user organizations in a series of digital workshops exploring how implementation-level design can be organized to better engage rural health workers in the process. The project has resulted in a set of concrete challenges and opportunities relating to how projects are organized and to engaging with the rural health workers in requirement gathering, prototyping, and evaluation.

Conceptualizing DHIS2 as a design infrastructure for implementation-level design helped shed light on several aspects relevant for further inquiries in the design lab. A key question became how the design infrastructure could best support contextual implementation-level design with technical flexibility and knowledge resources. Based on this question, several master's students engaged in a variety of projects exploring the issue from different perspectives. For instance, two students explored challenges and opportunities for improving a set of newly introduced resources supporting the development of apps for DHIS2 by testing them in use in India and Tanzania. Following their findings, a second group of three master's students attempted to design and develop a web-component platform for sharing web components between the partner organizations, aiming to make custom app development more efficient (framed as DSR). Furthermore, four master's students explored the knowledge resources accompanying the newly introduced app development resources, which now have been subject to iterative improvement and evaluation for three years, spanning several master's thesis projects. The knowledge resources have been evaluated, both with developers from partners and with students in a master's course at the UiO, where the students build apps for DHIS2 as part of their coursework. The course has served as a lab for testing various app development resources for several projects in the design lab.

While several projects have explored challenges and potential remedies related to the technical flexibility of the design infrastructure, others have examined knowledge resources aimed at supporting and promoting the use of methods for contextual design. For instance, a group of students has studied how a design method toolkit as part of the DHIS2 design infrastructure can support contextual implementation-level design (framed as DSR). While some students have explored the format of such a toolkit, others have investigated some partners' practices and challenges to develop appropriate content for the toolkit in order to offer support that is best aligned with their existing practices and the concrete challenges they face (outsider CS and DSR).

Several projects initiated in the fall of 2021 with new groups of master's students are defined as continuations of the previous efforts. One of these involves exploring capacity-building resources for learning how to utilize the API of the DHIS2 core solution to build custom apps (framed as a DSR project). Another one involves diagnosing challenges related to contextual design in a specific implementation project in Mozambique, in collaboration with the local partner there, and identifying ways of strengthening the contextual design in the future design process of the project. This is planned and organized as AR. The lessons learned from the project are aimed to be useful for the Mozambique partner and to feed into the content of the design method toolkit to be useful across partners.

In addition to the inquiries conducted by different projects, a set of activities seeks to continuously share the findings among the different projects and to sustain a shared and overarching set of problem formulations. For instance, the students regularly present and discuss their findings with the students involved in other projects and, when relevant, with core team representatives. Figure 4.7 offers some pictures of different activities of the design lab, including inquiries in Tanzania and Mozambique and discussions of the findings in Oslo, and a photo of our office.

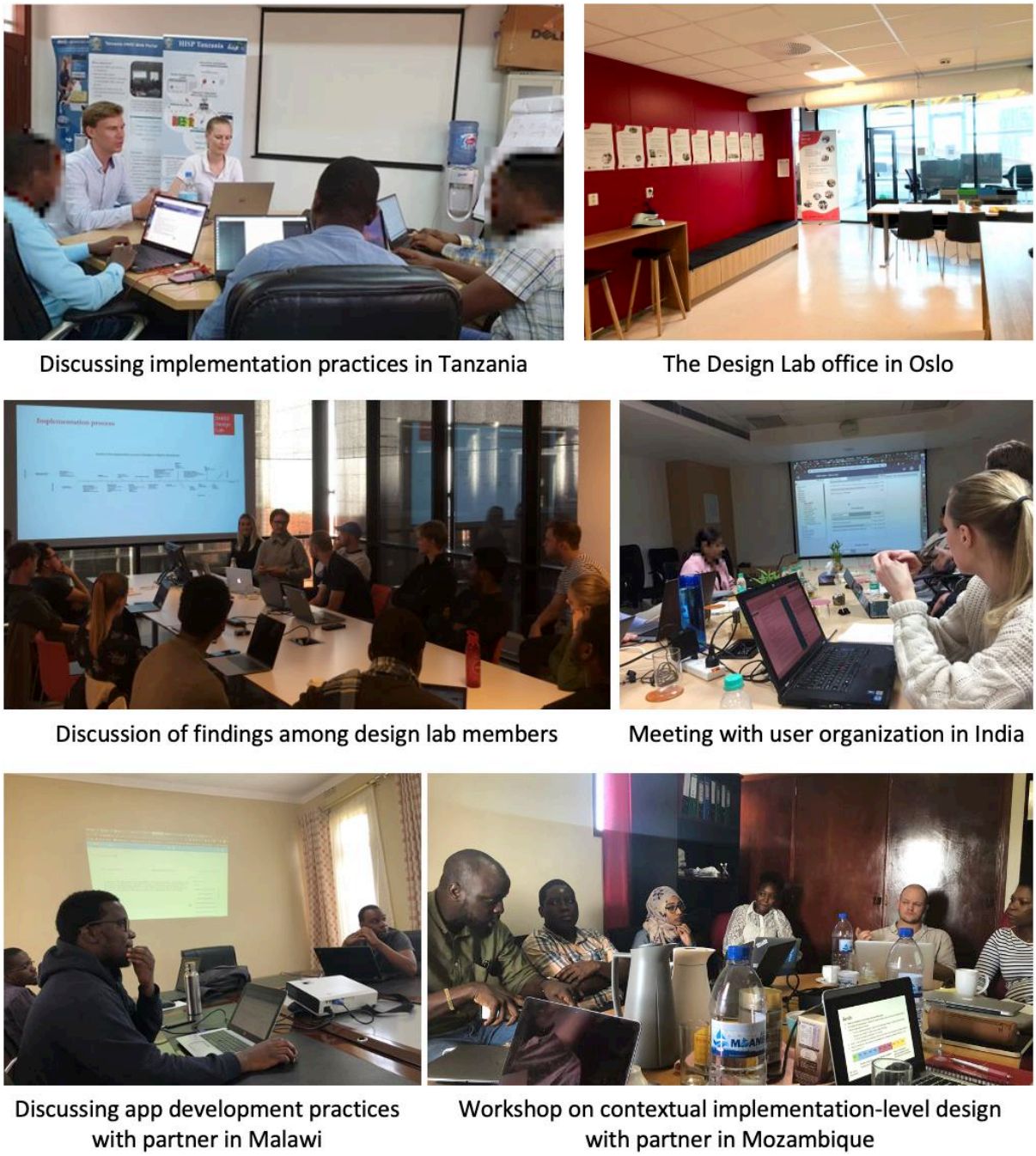

Figure 4.7. Pictures of some activities and locations of the design lab's work.

The design lab is still an ongoing project and has involved 46 master's students, 20 of whom are still actively involved in projects. The students have worked and are working within nine clusters of projects, focusing on aspects ranging from the core team's generic-level design practices to the partners' implementation-level design practices, and exploring resources for the design infrastructure. Table 4.3 summarizes the main clusters of projects in the lab and how they relate to the different elements of ES design as conceptualized by the theoretical framework. More information about the design lab and some of the ongoing projects can be found at:

https://www.mn.uio.no/hisp/english/dhis2-design-lab/.

Table 4.3. Main clusters of projects in the DHIS2 Design Lab.

Overall focus	Project clusters	Forms of inquiry
Implementation-level design	Exploring existing practices and how to strengthen contextual and user-oriented design practices with partner in India	Insider CS
	Exploring existing practices and how to strengthen contextual and user-oriented design practices with a broad set of partners	Outsider CS
	Exploring past experiences in a specific implementation project in Mozambique for intervention and evaluation to strengthen contextual design in the future design process	AR
Design infrastructure	Testing app development resources with partners	Insider CS, ADR
	Designing and evaluating component platform for efficient app development	DSR
	Designing and evaluating capacity-building resources for app development	DSR, ADR
	Designing and evaluating resources to support and promote contextual and user-oriented implementation-level design (design method toolkit)	DSR
Generic-level design	Exploring the core team's requirement collection and definition, practices, and challenges	Outsider CS

4.5 The Design Lab as an Approach to Studying ES Design

Based on the outline of my approach to engaged scholarship and the story of the design lab, I summarize and offer an explicit answer to RQ 2 on how researchers can study ES design in collaboration with relevant practitioners.

The brief story shows how the design lab has evolved to focus on the study of design processes (generic-level and implementation-level design) and a form of meta-design products of the design infrastructure, aiming to support implementation-level design processes. In doing so, the lab serves two roles: 1) as an engaged research engine where we combine different forms of engaged inquiry in sequence and in parallel, feeding into an overarching shared set of problem formulations, and 2) as an intervention in the DHIS2 ecosystem itself. Figure 4.8 illustrates how the lab is positioned in relation to

the key design processes and resources in the DHIS2 ecosystem using the theoretical framework of the thesis.

Characterized by an evolving understanding of the problem situation and opportunities for collaboration with relevant practitioners, using engaged scholarship as a methodological basis has proven useful. Rather than committing to a single form of inquiry (e.g., AR) upfront, it has allowed us to flexibly organize both the overall research approach of the lab and the various subprojects by selecting forms of inquiry based on their relevance and feasibility, given our dynamic relations with the real-world problem situation and the practitioners.

The theoretical framework has been developed, has guided, and has been elaborated through the various projects of the design lab. For instance, the projects exploring existing and new resources for the design infrastructure are based on the idea of a design infrastructure supporting implementation-level design and help develop a better understanding of the nature of a design infrastructure. Similarly, when initially defined, implementation-level design was an "empty" concept referring to a type of process. The further inquiries framed by it helped in understanding its nature and role in ES design. The framework has thus served as a common *kernel theory* (Jones & Gregor, 2007) for the lab, while a variety of additional concepts have been employed to study more specific phenomena.

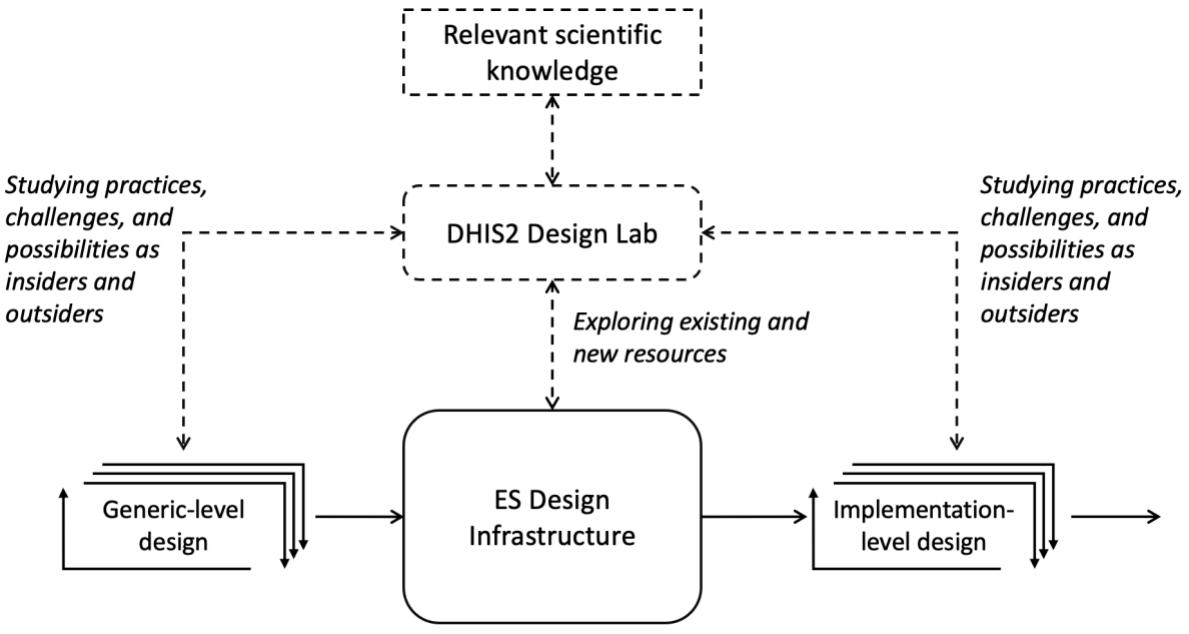

Figure 4.8. How the DHIS2 Design Lab is positioned to study the DHIS2 design.

To offer an explicit answer to RQ 2, the research approach outlined in the preceding sections suggests that ES design can be studied in collaboration with relevant practitioners by

- focusing on the study of processes of generic-level and implementation-level design and the nature and supportive capacity of the design infrastructure and
- using engaged scholarship as a methodological basis for diagnostic, design-oriented, and intervention-oriented research on processes of generic-level and implementation-level design and the resources of the design infrastructure. This allows collaborating with practitioners (e.g., the vendor and the partners) through several forms of inquiry for exploring current design practices and challenges (CS) and future resources (DSR and ADR), as well as introducing and evaluating changes to design practices (AR and ADR).

This could be organized as a design lab that involves multiple researchers and practitioners and combines several forms of inquiry in sequence and in parallel. In Chapter 6, I more broadly discuss how these findings from organizing the design lab contribute to research concerned with ES design and engaged scholarship in IS research.

4.6 My Roles and the Primary Data Sources of the Thesis

Moving from the overall aspects of the research project that also presented the findings related to RQ 2, I now elaborate on my roles and on the specific data sources underlying the findings related to RQ 1.

4.6.1 My roles

I have assumed many roles related to planning and carrying out the research project. I have established and headed the DHIS2 Design Lab, where I have had the overall responsibility of ensuring coherence among the different projects of the master's students, being actively involved in the planning, data collection, and analysis in each project. Based on the findings from preceding projects of the design lab, I have also been responsible for defining the new projects that take on new master's students. I have thus linked the findings of the different sets of inquiries and the new ones. Meanwhile, I have served as the primary or the secondary supervisor of many of the master's students. The design lab and I have also been part of the research group at the UiO, which has a close relationship with the core team as the vendor of DHIS2 and the partners. Furthermore, I have taught courses at the UiO, including the course that we have used as a lab to evaluate app-development resources for DHIS2.

A well-known challenge in engaged scholarship projects, and often discussed for AR in particular, lies in developing a well-functioning relationship with the practitioners (Simonsen, 2009). On one hand, I have worked to maintain my explicit role as a researcher to ensure that the master's students and I do not commit to too many practical

responsibilities that would make the project's research elements (e.g., methodological rigor, theory, academic contributions) be less prioritized. On the other hand, we want to ensure that we explore challenges that are well-grounded in the real-world problem situation and to contribute as much as possible back to the practitioners. For the relationship with the core team, this has developed throughout the research project through our collaboration on specific design lab projects and my participation in meetings and discussions. Through this, I have developed a causal relationship with some of the core team members, which has helped me and our research team in gaining access and being part of strategic discussions. Nonetheless, I have maintained my explicit role as a researcher and not assumed any formal roles in the core team. Similarly, in collaborations with partners, we have balanced between doing research *with* and *for* the practitioners (Van de Ven, 2007), managing expectations for the form and level of our contributions to helping with overcoming their challenges, while attempting to make up for the time and effort spent by the practitioners in working with us.

I have also been required to balance between being an independent researcher and supervising master's students while working with them in projects. While my work in developing theory, formulating overall problems for the lab, and defining new master's projects has guided the work of the students, their findings and work have fed back into these activities. In many of the projects highlighted above, I have participated in the data collection and analysis. Meanwhile, I have carried out data collection independently (outlined in the next section), and the analysis presented in this thesis has mainly been my own effort, in collaboration with my supervisor on the co-authored papers. To make this more independent work and the division between this and all the projects carried out with the master's students more explicit, I now highlight the primary data sources underlying the findings related to RQ 1.

4.6.2 Primary data sources of the thesis

Whereas the projects of the design lab have all influenced the research process, four primary data sources underlie the findings presented in relation to RQ 1. These are participation in the implementation project in India, visits to partners in Tanzania, Malawi, and Mozambique, online interviews with partners, and continuous dialogue, participant observation, and interviews with the core team. Figure 4.9 presents a timeline of the data collection.

First, a key source of data comes from participating in the implementation project in India (carried out as an insider CS). Here, I attended meetings with the user organization, participated in visits to health facilities, participated in internal planning meetings with the partner organization, and conducted a set of interviews and focus group discussions with the partner organization to understand their broader set of

practices and challenges, both related to the specific implementation project and beyond. These inquiries serve as the basis for Paper 1.

Second, the focus group discussions and interviews with the partners in Tanzania, Malawi, and Mozambique are key data sources for Papers 2 and 3 (carried out as an outsider CS). Guided by the theoretical framework developed from the engagement in India, the overall guiding questions for these focus groups were as follows: What is the potential for contextual implementation-level design? What are the challenges and obstacles? How could the DHIS2 design infrastructure better accommodate contextual implementation-level design? The visits lasted for about one week and involved focus group discussions, each taking about 2–4 hours for 3–5 consecutive days. The focus groups included developers, implementers, and the managers of the partner, and the sessions started with discussing the implementation projects in which they were engaged and examining their similarities and differences together. This helped identify the partner's general implementation practice (i.e., the typical activities it involved) and a set of general challenges. We then worked on outlining the typical implementation-level design process (in terms of activities) before discussing the various challenges of relevance to contextual design, linking them to concrete activities of the design process. At the end of the visit, I presented the thus far analyzed findings from the focus groups and the interviews to the participants for discussion.

Third, I conducted online interviews with a set of partners (from the USA, Uganda, Sri Lanka, and India) (outsider CS). The plan was initially to 1) be engaged more closely with the partner in Mozambique and potentially participate in one or several of their implementation projects, and 2) visit several partners in Africa (e.g., Kenya, Uganda, and South Africa). However, due to the COVID-19 pandemic, all travel plans were put on hold; accordingly, I decided to conduct four online interviews and otherwise invest my efforts in discussions with the core team (described below) and in supporting the projects of master's students exploring resources to strengthen the DHIS2 design infrastructure. Similar to the purpose of the visits to partners, the online interviews aimed at understanding the practices of implementation-level design and the challenges and possibilities for contextual design. Lasting 60–90 minutes each, the interviews were carried out as semi-structured discussions, often initiated by asking the participant to highlight the typical activities or steps involved in an implementation project. Using the overview of implementation-level design as a basis, I delved into specific activities and aspects of the process, discussing potential challenges related to contextual design.

Fourth, throughout the project, I have been in continuous communication with several key members of the core team, including four product managers (in charge of the design of specific modules of DHIS2), the lead UX designer, and two implementation specialists (offering support to partners in implementation projects). Some of this communication was related to the various projects that included master's students and

explored different aspects of generic-level and implementation-level design and the design infrastructure. These discussions were about understanding their perspectives on pertinent challenges (thus involving them in problem formulation) and communicating relevant findings that could aid them in their work. I also conducted 15 semi-structured interviews, focusing on the core team's practices and challenges in generic-level design, the team members' overall reflections on the challenges of accommodating a diverse set of user organizations, and their current challenges and future reflections on adopting a platform strategy. Finally, I participated in and observed a range of strategic meetings where the core team discussed challenges and reflected on the future evolution of DHIS2.

The data have been documented and analyzed concurrently with the data collection throughout the process. For the involvement in the implementation project in India, I took notes during all meetings and discussions, documented my own reflections on the process, and wrote daily summaries of key learnings. Many of the reflections related to surprises that triggered new questions and the development of temporary explanations for the observed challenges. Similarly, I took extensive notes during the focus group discussions, which I also complemented with a daily summary, attempting to answer predefined questions, reflect on surprises, and define new questions.

As mentioned, I also presented a summary of my preliminary analysis on the last day of the visits to the partners in Mozambique, Tanzania, and Malawi, bringing the participants into the analysis process where they corrected misconceptions and elaborated where things were missing. These summary sessions were not limited to repeating what the partners had explicitly stated in the focus group discussions but attempted to offer an analysis of practices and challenges beyond what was explicitly stated. The partners appreciated this, as it (in their own words) allowed them to see their practices in new ways and helped highlight areas of improvement. The benefit of having fewer habits related to the phenomena under study, in contrast to the participants, means that for good and bad, more of the observations lead to surprises and inquiry, helping the participants to question what may be perceived as given by them.

The summaries accordingly became valuable situations for further concurrent data collection and co-analysis. The interviews conducted with the partners (online) and the core team members (online and face-to-face) were recorded and transcribed and later subjected to coding and the development of themes guided by specific questions (elaborated below). For each transcribed interview, I also developed a summary with reflections on how the interviewee's experiences and statements related to the whole.

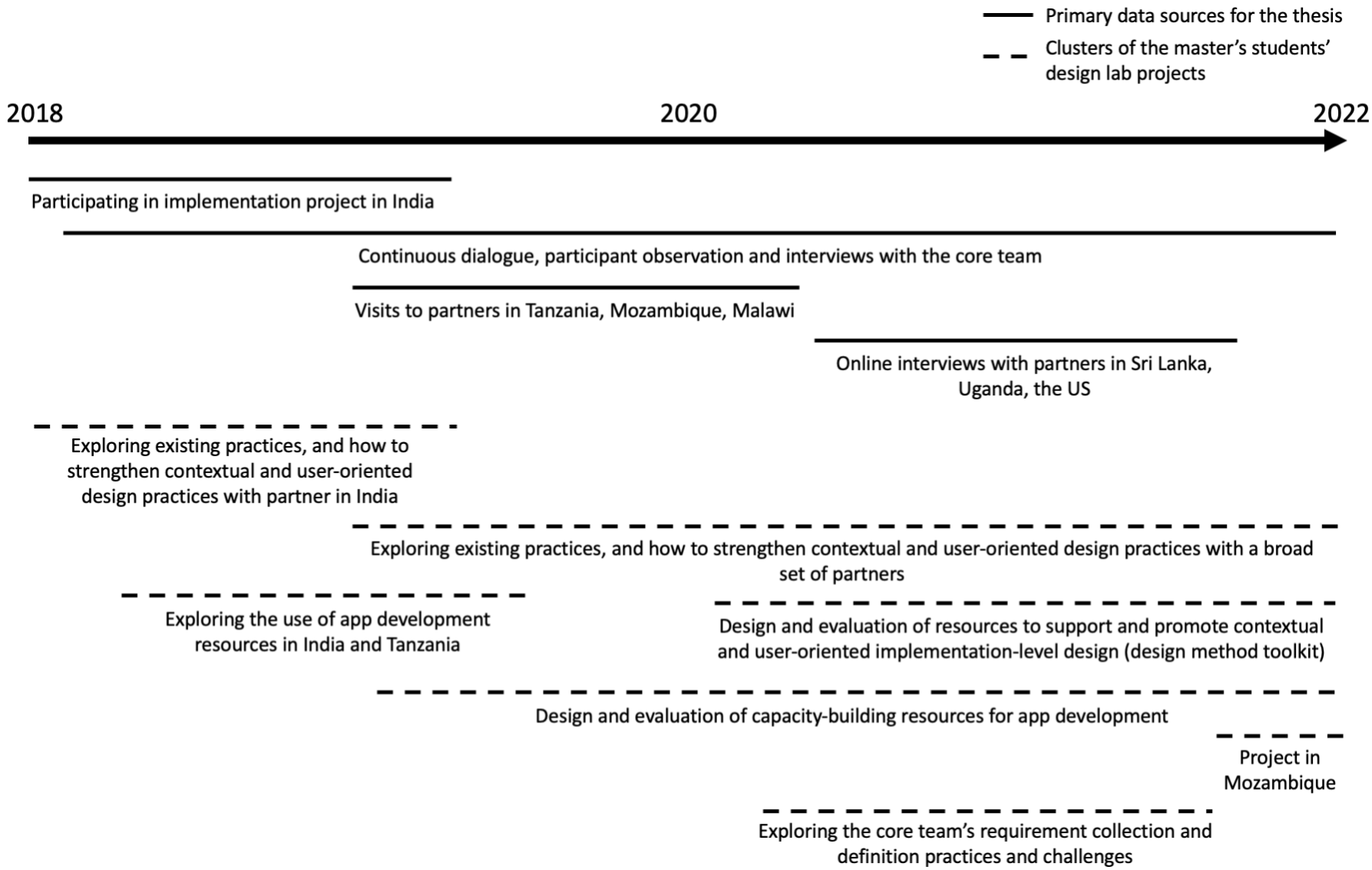

Figure 4.9. Timeline of the primary data collection of the thesis and clusters of the master's students design lab projects.

4.7 Data Analysis

I have used abductive analysis as an overarching approach to data analysis in the research project. Particularly, I have leaned on an approach to abductive analysis developed by Timmermans and Tavory (2012, 2014), which is based on Peirce's original concept and his work on pragmatism, and on Van de Ven's (2007) work on abduction in problem formulation and theory development in engaged scholarship. Abduction is a form of reasoning distinct from deduction and induction. It begins with an observation or a surprise, which leads to a search for theories and concepts that can help make sense of the problematic situation.

From the empirical observations of the problem situation, the researchers seek to establish an ongoing dialogue with existing conceptual schemes and insights from scientific knowledge. Initially, this includes a process of comparing the specific problem situation with a broad range of concepts and insights from academic literature. As argued by Timmermans and Tavory (2012, p. 169), in contrast to starting with a deductive or an inductive analysis, we are thus "neither theoretical atheists nor avowed monotheists, but informed theoretical agnostics."

The first step in making sense of a real-world problem situation is to identify concepts and insights related to similar situations that help make sense (e.g., describe and explain) of the specific problem situation under inquiry. A key question is thus what the problem situation is an example of. Doing so involves what Van de Ven refers to as an "abductive leap," which "is fundamentally an emptying operation in which the scholar strips away idiosyncratic details of the situation observed in reality. In doing so, he/she learns something generic about the problem that generalizes to a broader set or type of situations existing in reality" (2007, p. 105). In the process, the researcher experiments with different ways of "casing" the data, which "highlights different aspects of the phenomenon, rendering it comparable to different phenomena and turning it into a generalization that then can be linked to other fields and theories" (Timmermans & Tavory, 2012, p. 177).

For instance, DHIS2 and issues related to its design and implementation have been studied by researchers under conceptual schemes, such as open-source software development, participatory design, software platforms, and global public good. These are all valid abductive leaps, depending on the respective conceptual schemes that help "make one a more sensitive observer of [*the relevant*] details of action, better at asking useful questions, more capable of seeing the ways actions are patterned, and more adept at forming systemic hypotheses and entertaining alternatives" (Cronen, 2001, p. 30) related to the concrete problem in focus. Thus, on a very pragmatist basis, "the only criterion for useful inspiration is the general pragmatist guideline that theories are ways

either to ask new questions or to make new observations possible" (Timmermans & Tavory, 2012, p. 174).

My analysis thus began by identifying the phenomena examined in academic literature that have a) traits that are most similar to those of the real-world problem situation, where b) these traits comprise a significant part of the problem situation. The first exposure to the real-world problem situation occurred in the implementation project in India. Concurrent with the data collection were the experiments with different ways of casing the experiences with the problem, including concepts related to contextual design, usability maturity models, and participatory design. Over time, the empirical material highlighted the challenges that resonated well with concepts and insights from ES literature (e.g., misfits, customization challenges, generification, ecosystem). I thus identified generic ES as the most appropriate overall casing to understand the challenges faced in the DHIS2 context. Meanwhile, other dimensions of the phenomena have been casted using concepts and the associated academic literature related to contextual design and ES platform strategies, among others.

I now provide some details on the process of developing the theoretical framework, the conditions for contextual implementation-level design, and considerations for a platform strategy.

4.7.1 Developing the theoretical framework for understanding ES design

The development of the theoretical framework started during engagement in the implementation project in India. In the project, I tried to make sense of the problem of addressing usability issues by experimenting with different casings toward existing academic literature. Examples included usability design, limitations of usability maturity, and user-centered design. It was surprising to find how interdependent the design was during implementation with the design of the generic features they relied on, and that it made little sense to explore design methods without a better understanding of the surrounding context. The emerging issues showed how the context of implementation was highly dependent on the generic software features and their configurability. Understanding how the DHIS2 design could be strengthened thus had to take a systemic perspective. Accordingly, an aim was directed at framing *all* the design activities involved in shaping the software before it met the end users in order to better understand how we could approach addressing usability issues. The casing of the problem situation turned toward ES literature and thus to concepts such as misfits, customization, and generification. These helped describe the observed challenges but did not point to solutions. The meta-design concept proved helpful in understanding how the core team worked to support the design during implementation. Nonetheless, the literature lacked an overall conceptual scheme.

In Tavory and Timmermans' (2014) approach to abductive analysis, inductive and deductive cycles are intertwined with abduction. While experimenting with different concepts, the process of developing the theoretical framework has involved identifying patterns empirically, which are formulated as concepts and viewed in relation to existing scientific knowledge, as well as testing existing concepts from the literature to make sense of the empirical materials. These concepts that are informed by the empirical data also help in identifying patterns in the relevant academic literature. Thus, the process involves cycles of reading the related academic literature in light of the empirical findings and reading the empirical materials in light of the literature to develop the best possible understanding of the real-world problem situation. "Induction looks for the corroboration of generalizations, patterns, outliers, and salient themes in the data, while deduction suggests a reanalysis of existing data or new data-gathering rounds" (Timmermans & Tavory, 2012, p. 180).

The initial definitions of generic-level and implementation-level design and design infrastructure were therefore developed based on an amalgam of the empirical observations and existing ES concepts. The process involved coding and the development of themes from the field notes and transcriptions, while examining ES literature for similar patterns. Its grounding in existing literature is evident in Papers 1–3, as these all develop the framework from the literature, although its development was triggered by empirical observations of the real-world problem situation.

Since its initial definition from the engagement in India, and its presentation in Paper 1, the framework has helped frame further inquiries of the design lab and the data collection specific to the thesis project. Meanwhile, it has been continuously elaborated and refined in light of new surprises in the empirical materials, and the definitions of what comprises the *design infrastructure* and the nature of generic-level and implementation-level design have evolved. Most notably, while generic-level design was initially defined as an activity exclusive to the core team, the examination of the partners' increasing role in offering generic apps for the design infrastructure has led to redefining the concept.

4.7.2 Identifying conditions for implementation-level design

While we were working in India, it became evident that what I now refer to as *implementation-level design* represented a somewhat special and challenging context for contextual design compared with traditional bespoke software development. Interestingly, there seemed to be significant differences in how contextual and user-oriented various implementation projects and DHIS2 partners were. This triggered the inquiries into the other partners' practices and challenges related to contextual design. For the analysis, a broad body of literature on contextual design was consulted; however, it mainly focused on bespoke software development contexts (Sommerville,

2008; Van Fenema et al., 2007). A useful concept from the literature was that of boundary conditions (Zahlsen et al., 2020), which helped highlight the conditions around the design process. Whereas many of the observed challenges could be linked to how implementation-level design was dependent on the design infrastructure and the partners' practices, some other force seemed to be part of shaping the projects. I reviewed all materials from India, while the focus groups in Mozambique, Malawi, and Tanzania and the interviews were framed to directly explore the issue of contextual implementation-level design. Based on coding interview transcripts and field notes, I developed themes representing the conditions, which were iteratively sculpted by elaborating on, dividing, combining, and redefining the themes until they offered the best way of explaining divergence and convergence. Through this, the previously undefined force was identified as how the projects were configured in terms of scope, structure, and mandates, which I found to be interdependent with the partners' implementation practices and the design infrastructure. Table 4.4 offers an example of how statements led to codes, resulting in themes that finally represented the conditions and the relations between the conditions.

Table 4.4. Example of coding and sculpting of themes to identify challenges of and conditions for contextual implementation-level design (presented in Paper 2).

Data	Codes	Theme / condition
"Sometimes, they [the user organization] don't know what they want, but they know that they want something … this gives us more room to negotiate how the process should look like and for the solution to emerge over time." "All requirements were defined from the outset; we had no room for requirements to emerge during the process."	Project scope definition, negotiation of project with user organization	Project configuration
"I feel it is more important that the [team of researchers and the partner in India provide] support on the technical part – solving the problem rather than understanding more problems."	Limited appreciation of relevance of user-oriented design activities Partners' limited ability to convey benefits of user-oriented activities	Partners' implementation practice, relation between partners' implementation practice and project configuration

4.7.3 Identifying challenges and considerations for platform strategy

The theoretical framework highlighted meta-design as an important part of generic-level design and the core team's work in supporting implementation-level design.

Related to this, the design lab began to collaborate with the core team in understanding how we could better support app development on top of DHIS2. Meanwhile, the core team (as discussed further in the Findings chapter) sought to engage the partners in offering generic apps for DHIS2. The concepts of platform and platform strategy (Foerderer et al., 2019; Staub et al., 2021) helped make sense of these efforts by framing what the core team was trying to do and highlighting some key challenges that they were grappling with or could be anticipated to arise. However, the existing literature on platforms offered little insight into how a platform strategy would affect the dynamics of ES design highlighted by the theoretical framework. This led me to examine how implementation-level design would be affected by a platform strategy, particularly related to the processes' ability to address the specific needs of user organizations. To this end, I reexamined the materials from India and from the focus groups in Malawi, Tanzania, and Mozambique, which I again coded and developed themes from. Furthermore, a set of interviews with the core team members, specifically targeting the platform strategy, was carried out. The challenges were grouped into different types, and a set of associated considerations for the core team (and ES vendors adopting a platform strategy in general) was developed. These are presented in Paper 3. Table 4.5 offers an example of the move from the data to codes and themes.

Table 4.5. Example of coding and sculpting of themes to identify challenges of and considerations for an ES platform strategy (presented in Paper 3).

Data	**Codes**	**Theme**
"They [the partners] will of course prioritize their implementations when designing their generic apps. So that's a risk, and then we would have to maintain the old version [for the specific user organization]."	Using apps offered by partners, changing direction of generic-level design, dependability	Uncertainties of depending on the generic-level design of partners
"Our aim is that you can do the customizations you need and that these are compatible so that you can update all the other parts of the app that you didn't modify. Hence, you can stay connected and benefit from the ongoing development, without being totally disconnected to your own implementation."	Costs of customization, disconnecting from generic-level design, modularization, core team design strategy	Design and maintenance partitioning

With this overview of the research project, which offered findings related to RQ 2, as well as details on the modes of the data collection and analysis process, I next present the findings related to RQ 1.

5. Findings

I now turn to the findings related to RQ 1. First, I summarize my findings, categorized into three sets: 1) overall dynamics of ES design, 2) conditions for contextual implementation-level design, and 3) challenges of and considerations for adopting an ES platform strategy. I use the empirical cases presented in Papers 1–3 as the basis for the summary and complement these with additional empirical materials from the data collected in the thesis project, primarily in the form of quotes that underlie and help illustrate key observations and arguments. Second, I summarize the findings by articulating an answer to RQ 1.

5.1 Overall Dynamics of ES Design

In the first set of findings, I examine some of the general traits of how DHIS2 is designed to accommodate a diverse set of user organizations. In doing so, I empirically substantiate the theoretical framework outlined in Chapter 3 and highlight some challenges and opportunities related to ES design.

5.1.1 Accommodating many user organizations through generic-level design

The DHIS2 core team designs and maintains the dominant portion of generic software features for the DHIS2 design infrastructure. This includes a core comprising the generic data model and a set of core apps, covering functionality and UIs supporting widely relevant user needs, such as entering data and presenting these in reports, graphs, maps, and other kinds of visualizations. Together, the core and the apps make up the *DHIS2 core solution*. The core and the apps are subject to the core team's continuous improvement in terms of security and bug fixes and the introduction of new generically relevant functionality and UI improvements. Thus, new versions of the core solution are released four times a year. To inform the generic-level design processes for the core solution, the core team sustains an intricate set of relations and arenas. For instance, a public requirement ticketing system called Jira serves as the backbone of the core team's design and development processes and is open for partners and user organizations to submit requests for new or improved features. The system also features an open roadmap that offers an overview of the features planned to be included in the future versions of the software. Furthermore, the core team frequently corresponds with representatives from different partners and involves them in discussions on new features and the design of the overall roadmap. Additionally, a public Q/A forum called Community of Practice where partners and user organizations discuss challenges and share experiences is monitored to identify frequent issues and hold open discussions on new features.

A key challenge for the core team lies in the diversity and the extent of requests for generic software features in the core DHIS2 solution. The core team aims to design generic software features to be relevant across all or most user organizations and is thus often dismissive of requests that are only applicable to one or a few implementations. As articulated by a core team product manager:

> […] all the requests we have in Jira represent something like five to seven years of development work for our team. We can obviously not address all of them, particularly the ones that are very specific to a user […]. Relying on us is just not apt for supporting agile development cycles within specific projects.

Echoing this from the perspective of implementation-level design, the partners report their difficulty in having the specific needs of the user organizations they serve addressed in the core generic software features. A Malawi implementer says, "We can't rely on the requirements of our projects to be addressed by [the core team]. We try to manage ourselves with the space we have to adapt [DHIS2] within the implementations."

Even when the requested features are judged to be of wider generic relevance by the core team and thus implemented in the core solution, it will often take too much time for the partners to serve their user organizations in a timely fashion. A Tanzania implementer explains, "We submit a Jira ticket and keep our fingers crossed. Then maybe in 4–5 versions, it will be available." With four annual releases, it may take a year before their request is reflected in the core solution.

Consequently, rather than envisioning DHIS2 as a ready-to-use software, the core team increasingly views the role of the core solution as a "fundament" or an "infrastructure to build on." This renders implementation-level design an important part of designing DHIS2 to accommodate specific user organizations' needs. As a Mozambique implementer articulates, "Earlier, the [core team] was communicating more directly with [a large user organization] to address its needs in the generic DHIS2. Now, we have a much larger role in trying to address its needs locally."

To support implementation-level design, a significant part of the core team's generic-level design involves embedding flexibility, in terms of configurability and extensibility, in the core solution. A core team product manager says, "Thinking generic often involves thinking in terms of what must be configurable for the software to fit anyone." Configuration facilities are built to respond to the anticipated diversity that the core team considers necessary to be accommodated. Examples are organizational hierarchies, what data to be reported when and by whom, and how to present these data. In contrast to configurability, which is offered through standardized settings in the core solution, extensibility represents a generative environment, offering the potential to build radically new and unanticipated functionality and UIs. This is supported by

offering APIs and software development kits (SDKs). With increased flexibility, implementation-level design requires significant competence to successfully configure the core solution according to a specific user organization's needs. The core team offers a variety of knowledge resources that accompany the configurability and extensibility to develop the capabilities of those that engage in implementation-level design. This includes documentation, implementation guidelines and the community of practice (CoP). In addition, the core team and some of the partners organize regional learning and certification events called the DHIS2 Academies several times a year. The academies feature topics such as fundamental DHIS2 setup and configuration, how to implement DHIS2 to serve various types of use cases such as disease surveillance or health logistics management, the implementation of specific modules such as the DHIS2 tracker module, or app development for DHIS2. When graduating the one-to-two-week program, participants get a certification in the respective topic. The academies thus play a central role in building capacity for implementation-level design of DHIS2.

5.1.2 Accommodating specific user organizations through implementation-level design

Implementation-level design is essentially about addressing the needs of a specific user organization based on the generic software features of the DHIS2 design infrastructure. The process often begins with a call for tenders where a partner submits an initial proposal on how to address the needs and requirements outlined by a user organization. After a partner is awarded a contract, implementation-level design involves activities such as requirement gathering, prototyping, evaluations, end-user training, and roll-out. Another important activity is (re)negotiating the scope, structure, and process of the implementation project and the partner's mandate therein. To be competitive in their initial bids on the tender and to remain on budget throughout the implementation process, the partners seek to rely on generic software features and remain connected to the associated processes of generic-level design to benefit from continuous improvements and upgrades. Accordingly, the core solution plays an important formative role in all activities of implementation-level design, and the partners use the core solution as the starting point in initial discussions. A Mozambique implementer states that this "allows us to put up a prototype very quickly, often to the surprise of the managers [of the user organization]." In the proceeding activities, the prototype based on configuring the core solution is iteratively refined with several evaluations with the user organization.

For every need or requirement, the partners have four options on how to address each of them (summarized in Table 5.1). First, configuring generic software features is the preferred option, as this involves no development costs and allows staying connected to the associated generic-level design process to benefit from future updates. The

configurability of the core solution mainly relates to meta-data, that is, defining what data can be reported, by whom, when, and how these data should be presented in reports and data visualizations, such as maps and graphs. Figure 5.1 illustrates the configuration interface of the DHIS2 core solution. Second, the development of custom apps offers the flexibility to design novel UIs and functionality, at the cost of developing and maintaining them individually for the user organization. Many partners thus try to avoid custom app development as far as possible. Third, the partners may request features to be included in the core solution. Finally, many needs end up in negotiations to be curbed into something readily possible with generic software features, or if not considered high-priority needs, they are omitted.

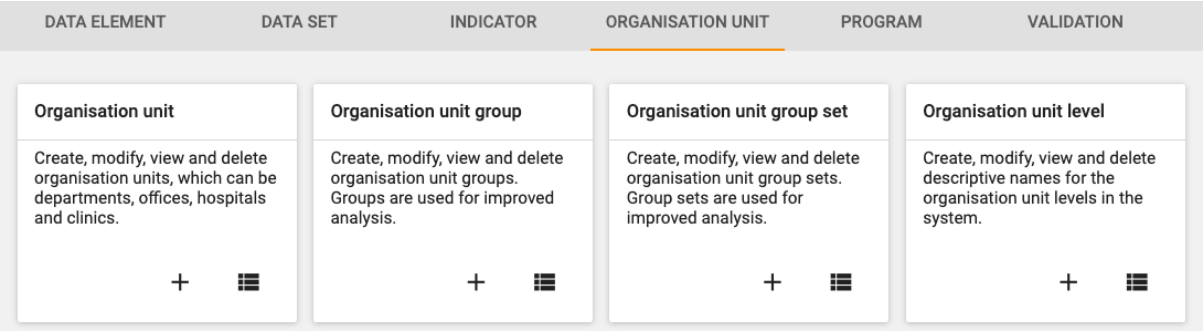

Figure 5.1. Example of screen for configuring the DHIS2 core solution during implementation-level design.

Table 5.1. The possibilities for addressing the specific needs of user organizations during implementation-level design, with pros and cons, according to the partners.

Option	Pros (+) and cons (–)
Configuration of generic software features	+ Low immediate and long-term costs for the user organization – Limited to the anticipated and designed-for diversity by the core team, mainly concerning the definition of meta-data
Developing custom app	+ Flexibility to design novel functionality and user interfaces – Immediate and long-term costs of development and maintenance are shouldered by the user organization
Request for features in core solution	+ Development and maintenance costs are covered by the core team – Unlikely that features are added by being considered too specific; it may take months or years before available in the core solution
Curb or omit the need	+ Avoid costly custom app development, may instead rely on available generic software features or on adapting the user organization's practice – May introduce usability issues, higher training costs, user resistance, less support from the software for needs considered important, and missed opportunities for digital innovation based on specific user needs

The relationships between the DHIS2 partners and the user organization commonly last for years and even decades. After an initial solution is rolled out, the partners may thus play the important role of monitoring and matching the evolving needs of the user organization with the evolving possibilities afforded by the DHIS2 design infrastructure. For the partners, it accordingly involves a two-sided monitoring process, requiring them to be cognizant of both emerging user needs and emerging technological possibilities. A (new) possibility that lies in the generic software features of the design infrastructure (e.g., a new data presentation app) may highlight an opportunity for improvement and spawn the user organization's interest. Similarly, an emerging need in the user organization may prompt a search for suitable generic software features or the development of custom apps to address it. As the needs of the user organization *and* the possibilities of the design infrastructure may evolve, interestingly, this also means that misfits between needs and software features may emerge, not only at the point of initial implementation, but also as the generic software features are subject to updates and refinements in subsequent versions. Furthermore, changes in the user organization's practices and needs may be incompatible with existing or new versions. Similar to the situation in the initial implementation, this may trigger the need for

custom app development or the curbing of needs at any time during the long-term partnership.

Several of the resources provided by the core team support partners in remaining cognizant of the evolving possibilities that lie in the design infrastructure. For instance, the partners are encouraged to regularly attend DHIS2 academies to stay up to date on new features. The Community of Practice also offers an arena to follow updates from both the core team and the wider ecosystem in terms of challenges, experiences from implementation projects, discussions on the challenges, and new features.

Summary of the first set of findings

- As the core team members grapple with the diversity and the large number of requests for the DHIS2 core solution in their generic-level design efforts, they increasingly emphasize *meta-design*, offering configuration and extension facilities to support implementation-level design.
- Processes of implementation-level design are in turn dependent on the designed-for flexibility provided by generic-level design to configure and extend the generic software features according to the specific needs of user organizations.
- Accordingly, DHIS2 is designed to accommodate a diverse set of user organizations through processes of generic-level and implementation-level design working in tandem.
- To keep costs low, the partners aim to dominantly leverage generic software features and remain connected to their associated processes of generic-level design to benefit from continuous improvements and updates.
- The ability to address specific user needs during implementation-level design is not merely a function of flexibility but also of how *affordable* it is to utilize the flexibility in terms of immediate and long-term costs of development and maintenance.
- Implementation-level design processes often involve long-term partnerships between the partner and the user organization.
- The partner stands between two evolving systems: the design infrastructure and the user organization.
- The partner monitors and tries to match opportunities arising as new generic software features are made available or as new needs arise within the user organization.

5.2 Conditions for Contextual Implementation-Level Design

Acknowledging that the DHIS2 core solution "cannot be everything to everyone" (core team product manager), the core team wants to promote contextual implementation-level design, and the configurability and extensibility of DHIS2 are designed to accommodate this. The core team's motivation is twofold: First, with their limited ability to sustain direct and intimate relationships with most implementation projects, they want to ensure the usability and relevance of DHIS2 *through* the implementation work carried out by the partners. Second, they perceive implementation-level design as a potential context for relevant digital innovations to emerge, being closer to the context of use. These innovations could potentially inform the design of the core solution, and increased emphasis on contextual design activities might be an engine for such innovations to emerge.

To reiterate the information presented in Chapter 3, contextual design emphasizes the design of technology based on the end users' practices and needs in specific contexts. To do so, activities such as ethnographic inquiries into the context of use in order to study user practices and iterative prototyping and involvement of end users in evaluating these are employed to inform technology design. While studying the practices and challenges of the partners conducting implementation-level design, I observed significant differences in contextualness and user orientation between partners and projects. For the second set of findings of this thesis, I identify three conditions affecting the potential for contextual implementation-level design, with implications for ES vendors seeking to support and promote it. These are detailed in Paper 2 and illustrated in Figure 5.2; the following sections offers a summary of each.

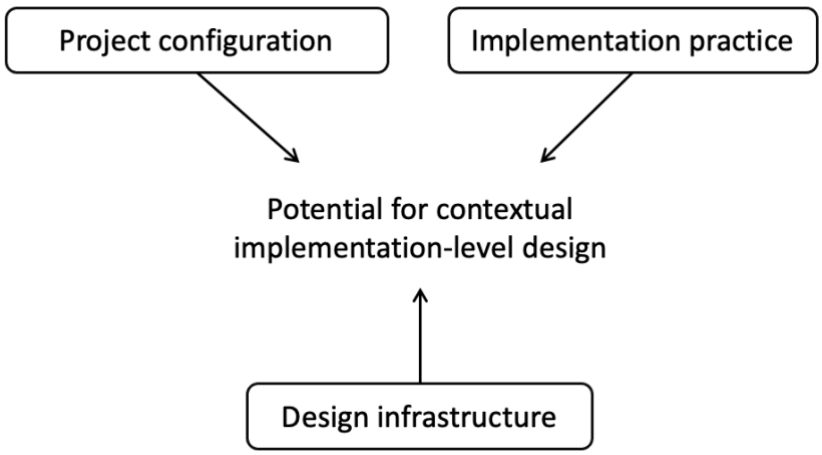

Figure 5.2. Three conditions for contextual implementation-level design.

5.2.1 Condition 1: The project configuration

An important part of implementation-level design is the meta-activity of (re)negotiating the scope and structure of the implementation project and the partner's mandate therein. There are significant differences in how projects are configured, which have significant impacts on the potential for contextual design. For instance, some projects start with a detailed list of more or less finalized requirements, and the partners may act only as technical consultants, mainly tasked with materializing the requirements into software features. In other projects, objectives may be defined in terms of some overall goals for improvement, where the partners have significant room for identifying the most appropriate means of attaining these goals with technology. The latter type of project configuration is far more fertile for contextual design than the former, and the partners have a greater ability to influence how the requirements are defined (e.g., based on an in-depth inquiry into end-user practices and user involvement rather than assumptions of IT managers). The former allows little room for the IT solution to evolve, based on iterations of prototyping and evaluation, because when "all requirements [are] defined from the outset, we [have] no room for requirements to emerge during the process" (Mozambique implementer).

A challenge in configuring projects for both the partner and the user organization is to balance the flexibility and the predictability of the implementation project. On one hand, flexibility in terms of aims and scope is needed to allow technical solutions to emerge, based on the knowledge of the end users' practices and needs during the design process. On the other hand, both the user organization and the partner value some level of predictability for the project. For the user organization, it must have some assurance on the outcome of its investment; for the partner, there is the fear that the user organization will misuse flexibility to "constantly change and expand the scope" (Indian implementer).

How projects are configured also affects what amounts to affordable design flexibility in the project. As highlighted in Section 5.2, designing custom apps offers flexibility for building novel functionality and UIs. However, the development and maintenance costs fall on the user organization. Accordingly, whether the project is configured to cover costs related to custom app development determines the extent of contextual design that is relevant. If the project is expected to rely only on configurability, aspects of meta-data are at the center. However, when the design of custom apps is part of the partners' mandate, in the words of a Tanzania implementer, it "forces us to look at other aspects, such as the kinds of layouts and functionality that will best support the user."

5.2.2 Condition 2: Partners' implementation practices

There are also significant differences in the partners' practices related to undertaking contextual design. For instance, some partners perceive contextual design activities,

such as ethnographic inquiries into the context of use and end-user involvement in evaluating prototypes, as burdens that ideally, should be avoided. These partners typically describe their aim of interactions with users as somewhat of a "user signoff" – obtaining the user's (quick) approval to proceed with the choices made. In contrast, representing the view of other partners, the Mozambique implementer argues, "If you're not doing it [field visits and end-user involvement], you go in blind [to the development process] […]; you need to understand the context." Accordingly, they employ several methods of understanding the user's context and evaluating prototypes for relevance and usability. Furthermore, the necessity and the benefits of attending to end-user practices and needs as part of the implementation-level design process are not always apparent for the IT project managers of the user organizations. Thus, the partner plays an important role in negotiating contextual design activities in the project configuration. Again, there are significant differences in the partners' practices. Some, such as in Mozambique and Malawi, view their role as "fighting the battle on behalf of the end users" (Mozambique implementer) and work actively to configure projects that are susceptive to requirements emerging from contextual design activities. In Malawi, for example, they negotiate for methods, such as user-centered design, to be an explicit part of the project structure. Others only include contextual design activities in projects when demanded by the user organization.

5.2.3 Condition 3: The formative effect of the design infrastructure

While there are many differences in how projects are configured and in the partners' practices and attitudes regarding contextual design, there are also some striking similarities in the practices of implementation-level design. I argue that these are largely due to the nature of generic software features, their adaptation capabilities (configurability and extensibility), and the knowledge resources of the DHIS2 design infrastructure. As shown in the first set of findings, the configuration facilities of the DHIS2 core solution are predominantly geared toward supporting the definitions of the data elements to be reported and how these are presented in reports, graphs, and maps. This is further reinforced by the knowledge resources of the design infrastructure. The implementation guidelines and educational materials presented at the DHIS2 academies naturally promote a design process that is aligned with this "data in–data out" focus. Overall, this amounts to the promotion of a highly standardized implementation-level design process. Interestingly, what is readily configurable in the core solution and the implementation practices promoted by the knowledge resources of the design infrastructure seems to manifest itself in the partners' implementation practices, in how the projects are configured, and thus, in the design process itself.

During requirement gathering, for instance, what the implementers look for is closely aligned with what is readily configurable in the core solution. An Indian implementer explains the essence of their implementation-level design process, which is very similar

among the partners: “We identify the data elements, then we try to implement the data outputs and present them to the client […]. We iterate between input and output several times.” Along the same lines, the Mozambique partner lead explains that in most projects, their primary focus is on “digitizing” existing paper-based reporting regimes (project scope) and thus on what data have to be reported and how these should be presented, which are “accommodated very well in the [core solution].” However, in projects that go beyond this scope, where the development of custom apps is part of their mandate, they have a broader focus. He explains, “the way DHIS2 is designed now [with the possibility to build custom apps] has allowed us to focus on these other aspects [functionality and UI]. Previously, this was not possible to change.”

The design infrastructure and the project configuration thus play a role together in shaping what kind of contextual design focus is relevant for the project, for instance, during requirement gathering and prototyping. Meanwhile, the project configuration is itself highly influenced by the design infrastructure since the generic software features often serve as a basis for the initial scoping and structuring of the implementation project. The nature of the design infrastructure also seems to be manifested in the experienced implementers’ practice of configuring projects and in carrying out implementation-level design. In sum, the design infrastructure exerts a formative effect on the focus of design and innovation during implementation-level design in certain directions. Aligned with the configurability of the generic software features, the focus is primarily on data in–data out, while the aspects that would invite changes to UIs and innovations in functionality may receive less attention.

The implementers’ cognizance of the design infrastructure is interesting. On one hand, it is necessary to plan and carry out realistic projects that best utilize the generic software features of the DHIS2 design infrastructure while retaining low costs for the individual user organization. Experienced implementers have thus internalized what is promoted by the design infrastructure. Meanwhile, inexperienced junior implementers are less colored by this lens:

> It’s a challenge when we send in the junior implementers first, and they promise *everything*, often many ideas that are interesting and useful but are expensive. Then, later, we [the seniors] have to come in and reduce the ambitions according to what can be done; it’s like a good cop, bad cop situation. (Tanzania implementer)

On the other hand, as a downside, the design infrastructure seems to gravitate project configurations and implementation practices toward having a blind-spot to aspects beyond the data in–data out configuration.

This formative effect is recognized as a potential challenge by some of the core team members, as expressed by a core team product manager:

> By having a lot of end-user functionality and by trying to make all this [configurability], we are making [it] so the [core solution] can do a lot. It's pushing out [the initiative to] innovate something to address specific user requirements because the core gives you 80%, so the cost–benefit analysis that comes at that last 20% is not worth the cost of doing it, then we're just using the core. But then, you lose that last 20% of end-user acceptance that goes a long way in institutionalizing it in the user organization.

Figure 5.3 illustrates how the design infrastructure exerts a formative effect directly and indirectly on the process of implementation-level design. It directly affects the implementation-level design process by acting as a lens during activities, such as requirement gathering and prototyping. It also influences how projects are configured and shapes the practices of the partners who seem to have institutionalized ways of negotiating and carrying out implementation-level design in line with the generic software features and the knowledge resources.

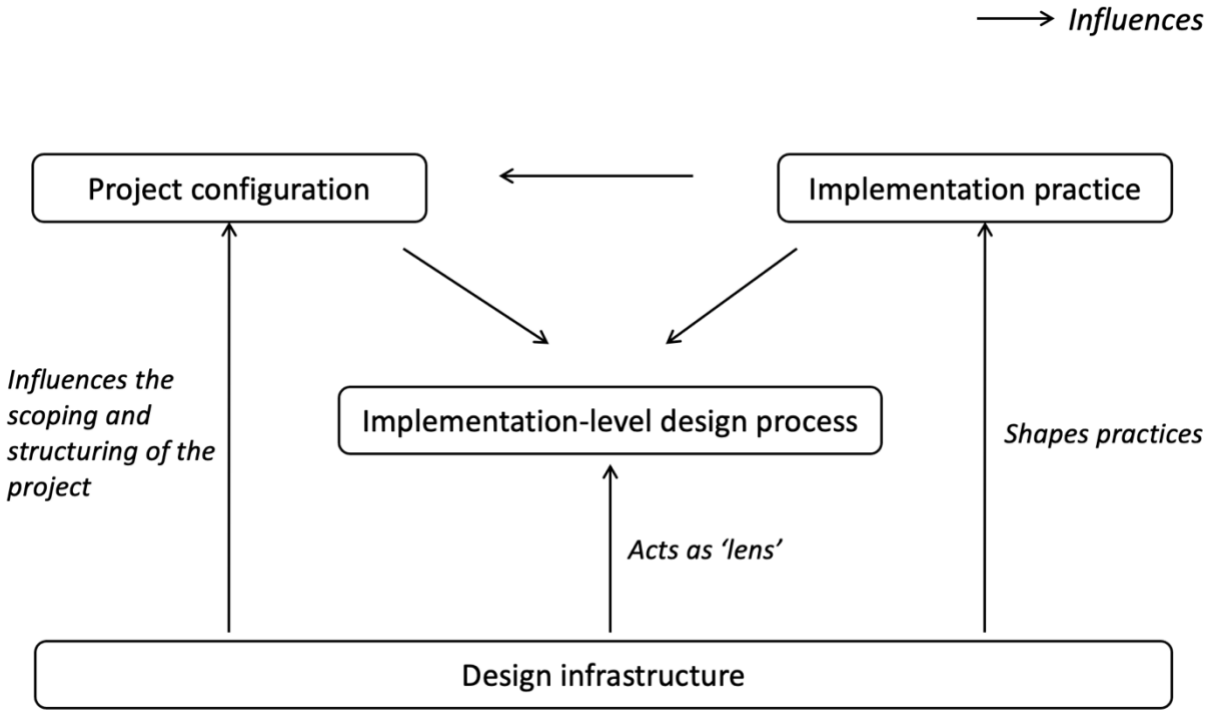

Figure 5.3. The design infrastructure influences the implementation-level design process both directly and indirectly via the project configuration and the partners' implementation practices.

Summary of the second set of findings

- With the flexibility afforded by configurability and extensibility, implementation-level design is a relevant context for contextual design and innovation to accommodate the specific needs of user organizations.
- However, three conditions affect the potential for contextual implementation-level design: the project configuration, the partners' implementation practices, and the design infrastructure.
- The design infrastructure exerts a formative effect on implementation-level design.
- The formative effect is manifested in the partners' continuous effort to "think in terms of generic DHIS2 features and map requirements according to these" (Indian implementer).
- The formative effect of the design infrastructure is driven by the rationale of retaining low development and maintenance costs during implementation-level design by leveraging generic software features without disconnecting from their associated processes of generic-level design.
- The knowledge resources of the design infrastructure appear to reinforce the effect by promoting standardized ways of undertaking implementation-level design, which aligns with the configurability of the generic software features.
- The formative effect pulls directly on the design process, as the generic software features act as lenses in activities such as requirement gathering and prototyping.
- The effect also pulls indirectly on the design process by having a formative effect on how projects are configured and the partners' implementation practices.

5.3 Challenges of and Considerations for Adopting a Platform Strategy

The final set of findings concerns the core team's ongoing efforts in turning DHIS2 into (in the core team's words), "a real platform." In general, the rationale for following such an approach is to better accommodate the diverse set of user organizations. Nonetheless, the notion of a real platform bears different meanings among the core team members. Some think of a platform as a type of software that serves as the basis for customization. Thus, becoming a real platform involves improving the extensibility of the DHIS2 core to allow less costly development of custom apps during implementation-level design. However, the main strategists behind the DHIS2 platform approach have the idea of a real platform that is more in line with the conceptualization presented in IS literature examined in Chapter 2: that of outsourcing parts of the design and maintenance of

generic software features (in the form of apps) to partners. Thus, it follows a model similar to that of consumer software platforms, such as iOS and Android, and prominent ES vendors, such as SAP and Salesforce.

Thus, the core team has a twofold aim: to better support the design and development of custom apps during implementation-level design and to encourage and support partners in developing, maintaining, and distributing generic software features that are made part of the design infrastructure. Both rationales are related to the aim of better accommodating the specific needs of the diverse set of user organizations. The former does so by supporting more flexible implementation-level design, and the latter engages partners in extending the portfolio of generic software features available for implementation-level design. However, achieving these aims is not a straightforward process. In Paper 3, my co-author Petter Nielsen and I identify three key considerations for ES vendors adopting a platform strategy. These are identified and elaborated based on the insights into the dynamics of ES design and implementation-level design in particular. The considerations are design and maintenance partitioning, dependability of generic software features, and design infrastructure cognizance and navigability. These are summarized in Table 5.2 (from Paper 3), elaborated more extensively in Paper 3, and outlined in the following sections.

Table 5.2. Considerations for ES vendors adopting a platform strategy (from Paper 3).

Title	Consideration
Design and maintenance partitioning	Partition design and maintenance between actors and processes to best provide flexibility to address specific user needs while sharing most of the development and maintenance costs between implementations
Dependability of generic software features	Address uncertainties tied to the continuity and future direction of design of generic software features offered by partners
Design infrastructure cognizance and navigability	Support partners in remaining cognizant of and in navigating an evolving set of heterogeneous generic software features provided by both the platform owner and the partners

5.3.1 Consideration 1: Design and maintenance partitioning

Two basic elements of the core team's platform strategy involve adopting a modular platform architecture for the core solution and offering APIs and SDKs to support the development of apps on top of the core solution. The platform architecture and extensibility resources offer an immediate benefit; it allows the design and development of custom apps during implementation-level design. In contrast to the configurability

embedded in the core solution, custom apps afford flexibility for design beyond what is explicitly anticipated and designed for by the core team. It also means that when choosing to customize a generic app, the implementation is only disconnected from the ongoing design and maintenance on that particular app, not the entire core solution. However, as discussed, custom app development is considered expensive in terms of development and maintenance and is accordingly often avoided. Furthermore, issues that would warrant custom design beyond what is supported with configurability might not concern only a single app but could permeate the whole core solution.

To strengthen the *affordable* design flexibility, the core team is experimenting with different measures. For instance, the team is currently exploring the possibility of "in-app extensibility" by designing some of the individual core apps as collections of widgets. During implementation-level design, different widgets can be selected and arranged according to specific needs, and new widgets can be designed. The cost of maintaining such a custom widget is far less than that of an entire app. As explained by a core team project manager:

> Tracker [generic core app] is very extensive, almost a platform in itself. In addition to the front-end, it has a large core where a lot of advanced stuff happens. [Even] if you only want to modify the user interface, you have to deal with the whole thing. […] [Now,] we're working on extensibility within the Tracker app. You plug in code here and there to change something, yet you avoid maintaining the code for the whole thing – you can still upgrade the rest of the app.

Allowing in-app widgets renders each app a platform in itself. An open question is how broad these widgets should be to offer the most relevant and affordable flexibility for implementation-level design. The second intervention is to offer a component library as part of the SDKs. The components are pieces of functionality and UIs that often surface in custom apps, which can be assembled in different ways during implementation-level design yet remain maintained by the core team.

The second important question for the core team is in what form the partners can best contribute to designing and sustaining generic software features. Currently, they do so in terms of full-sized apps, which are distributed on the DHIS2 *App Hub*, which is an app marketplace where partners and other third parties can upload and distribute generic apps. However, it is costly to design and maintain generic apps, "maybe multiplied by ten as compared to making a custom app" (core team product manager). Committing to sustaining the app over time is also a risky endeavor for the partners (as will be explored more in the second consideration). This means that many of the apps offered by the partners are not sustained in the long run, implying a risk for adopters. The core team members' ongoing conversations tackle the question of whether the app widgets or the

SDK components are better forms for partners to contribute with generic software features. A core team product manager explains:

> With a component framework, the pieces are smaller – if you build a smaller component, then the chances are greater that it is sufficiently general to be used in other places. It is more likely than when building a larger thing such as a vaccine certificate, where it soon contains something very specific for [a specific user organization].

Accordingly, in its adoption of a platform strategy, an important consideration for the core team is how to partition design and maintenance between actors and processes to best provide flexibility to address specific user needs, while sharing most of the development and maintenance costs between implementations.

5.3.2 Consideration 2: Dependability of generic software features

An interesting challenge associated with opening up generic-level design for partners is that it introduces uncertainties related to the continuity and future direction of design for the generic software features.

First, as shown, immediate and long-term costs are vital concerns during implementation-level design. When using generic apps offered by other partners, a key question in implementation projects is "at the end of the day, who will be responsible for maintaining the app?" (core team product manager). Should partners and user organizations engaging in implementation-level design expect the generic apps offered by partners to be subject to the same level of sustainability as those offered by the core team? Alternatively, should they expect that at some point, further maintenance could be discontinued, thus introducing costs for maintenance work to be incurred by the individual user organizations?

Second, the generic apps offered by both the core team and the partners are not static but subject to continuous design with modified or new functionality and UIs. This may mean that a generic app that at one point is appropriate for a specific user organization later drifts toward something that is no longer in line with the organization's practices and needs. Misfits may thus not only occur during the initial implementation but at any point in time throughout the software's lifespan in the organization. In these situations, partners and user organizations may be forced to disconnect from the associated generic-level design process and take on further maintenance to retain the previous version as a custom app. As outlined in the first set of findings, the core team sustains a wide range of arrangements for allowing partners and user organizations to monitor and influence the roadmap of the generic features they offer. However, for generic apps offered by the partners, there is no formal and standardized way to monitor and influence the further direction of design and no indication regarding their continuity.

What makes this issue particularly pertinent for the generic apps designed and maintained by partners is that these apps often originate from specific implementation projects and thus spring from implementation-level design processes. The partners' underlying motivation is to reduce the costs of offering the specific app to a single user organization by making it relevant to several user organizations they serve. However, when the apps are adopted by other partners in their implementation projects, they risk having little priority in decisions of the future direction of design: "They [the partners] will, of course, prioritize their implementations when designing their generic apps. So that's a risk [changing the direction of design]." (Sri Lanka implementer)

If the partner offering the generic app decides to take the app in a direction incompatible with the needs of user organizations served by other partners, "we would have to maintain the old version [for the specific user organization]" (Sri Lanka implementer).

Addressing uncertainties of continuity and future direction of design may require the introduction of some rules to be imposed on those offering generic apps. However, the core team is hesitant for three reasons. First, it goes against the vision of retaining DHIS2 as an open ecosystem with minimal rules. Second, it may potentially imply greater governance costs for the core team. Third, it may make the commitment to offer generic apps too risky for partners, thus reducing the likelihood of someone investing in the endeavor. However, with the lack of rules and standards, the uncertainties tied to using generic apps offered by partners may render them unattractive during implementation-level design. An alternative approach to the rules (as discussed above) is to change the form of the generic software feature contribution made by partners from entire apps to smaller widgets or components. Another option is to explicitly promote generic apps offered by partners only as a basis for custom app development, ensuring no expectations regarding sustained generic-level design and maintenance. However, the latter alternative would severely limit the economic benefits of sharing the design and maintenance costs among user organizations through generic-level design.

How to best address uncertainties tied to the continuity and future direction of the design of generic software features offered by partners thus represents the second important consideration for the core team in adopting a platform strategy.

5.3.3 Consideration 3: Design infrastructure cognizance and navigability

Finally, with few rules and standards regarding the process and the product of partner-driven generic-level design, the partners use different technologies (e.g., programming languages and frameworks) in their apps. This has resulted in generic apps with various levels of quality (e.g., performance, security, and stability), which are not necessarily compatible with one another and with the apps offered by the core team.

On one hand, diversity in what and how the partners build generic apps is considered desirable by the core team, as expressed by a core team product manager:

> We are by and large ignoring in the [core solution] a lot of these cutting-edge innovations – it's a tradeoff, and right now, the trade has been toward […] performance and stability, and we've kind of shut the door to this rapid innovation move-to-fast-break-things […]. The [partners'] apps should be an arena to do those things.

On the other hand, from the perspective of implementation-level design, the differences in technology and the resulting compatibility and quality issues pose a challenge for the partners to remain cognizant of and navigate the design infrastructure.

Again, for the core team, this involves a balancing act between openness, control, and governance costs. However, the App Hub represents one arena where the core team attempts to better support the partners in navigating the generic apps. The core team also plans to feature more indicators of aspects such as quality, compatibility, and sustainability, as well as how to possibly influence and monitor the future direction of the apps. Additionally, the extension resources, such as the SDK and the components, represent an effort that helps in standardizing certain elements of apps. A core team product manager explains:

> One of the biggest successes but also challenges of the app platform specifically is that it actually restricts what you can do to some extent […]. We're actually taking away some of the freedom of the developer, and that actually helps homogenize and standardize the way that applications are built and the way that applications can interoperate as well.

In this regard, for the core team, the final important consideration related to its platform strategy is how to support partners in remaining cognizant of and navigating an evolving set of heterogeneous generic software features provided by both the platform owner and the partners.

Summary of the third set of findings

- Adopting a platform strategy to open up generic-level design for partner organizations introduces several challenges during implementation-level design.
- The challenges relate to the costs of deviating from generic-level design, uncertainties of depending on the generic software features offered by the partners, and issues with navigating the design infrastructure.
- Accordingly, important considerations for the core team when following this route are
  - how to partition design and maintenance work between processes of generic-level and implementation-level design and among themselves and the partners,
  - how to address the uncertainties of depending on the generic software features offered by the partners, and
  - how to ensure design infrastructure cognizance and navigability.

5.4 Answering RQ 1 of the Thesis

In this section, I summarize the findings with an explicit answer to RQ 1: How can ES be designed to accommodate the specific needs of a diverse set of user organizations?

The findings of the thesis suggest that a fruitful avenue for ES vendors is to see their primary work as to cultivate their ES solutions as design infrastructures supporting contextual implementation-level design. This means that generic-level design should emphasize not only the design of ready-to-use functionality and UIs but also adaptation capabilities, such as configurability and extensibility to support implementation-level design. An important concern during implementation-level design is to limit the user organization's costs of developing and maintaining custom software features. The adaptation capabilities must thus be designed to offer relevant flexibility for implementation-level design while retaining the dominant costs of development and maintenance on processes of generic-level design. Whereas standardized configuration facilities allow costs to remain allotted to generic-level design, their rigid nature may impede flexibility and creativity in the implementation-level design process. The empirical case shows that an alternative to configurability may be to sufficiently modularize generic software features so that implementation-level design has the flexibility to organize them into custom features and that extending them involves minimal immediate and long-term costs.

Given a certain degree of *affordable* design flexibility, implementation-level design is a potential context for contextual design to accommodate the specific needs of user organizations. However, how implementation projects are configured, the partners'

implementation practices, and the design infrastructure are conditions that affect this potential. As the dominant influential condition, the design infrastructure exerts a formative effect on the way that implementation projects are configured and the implementation practices of partners. Depending on the nature of the generic software features, their adaptation capabilities, and the knowledge resources, the design infrastructure may shape implementation projects toward more or less contextual implementation-level design. This is both a challenge and an opportunity for ES vendors. It is challenging because efforts to support and encourage contextual implementation-level design require an emphasis on how these three conditions play out together, as well as the influencing role of the design infrastructure. It is an opportunity as it means that vendors may strategically cultivate the design infrastructure toward promoting implementation-level design that focuses on aspects considered important to be addressed locally.

Finally, adopting a platform strategy to support and encourage partners to design and maintain generic software features that are made part of the design infrastructure can be beneficial as it helps expand the portfolio of generic software features during implementation. However, it introduces several challenges to implementation-level design, and for a platform strategy to help design ES that accommodates a diverse set of user organizations, ES vendors must consider the following:

a) How can they partition design and maintenance between actors and processes to best provide flexibility to address specific user needs, while sharing most of the development and maintenance costs between implementations? This includes what aspects of design should be addressed during implementation-level design to secure local usability and relevance and what represents the most viable form of the partners' generic software contribution, considering issues of continuity and predictability.
b) How can they address the uncertainties tied to the continuity and future direction of the design of generic software features offered by the partners? This involves identifying the resources and rules that balance between openness and control.
c) How can they support the partners in remaining cognizant of and navigating an evolving set of heterogeneous generic software features provided by both the platform owner and the partners? This can be achieved, possibly by introducing standards without limiting the potential for innovation.

In the next chapter, I discuss how my theoretical framework, findings, and research approach contribute to IS research and outline a concrete set of implications for practice.

6. Contributions

I now discuss the contributions of this thesis. As outlined in the Introduction chapter, the thesis offers three contributions related to RQ 1 and a methodological contribution based on my research approach for RQ 2. I explain each contribution in greater detail in the following sections before presenting a set of concrete implications for ES vendors, partners specializing in implementation, and user organizations.

6.1 A theoretical perspective on ES design

The first contribution of this thesis is the theoretical framework developed and applied through Papers 1–3, which serves as a basis for the consecutive contributions. Developed through an analysis of patterns in the relevant academic literature and the empirical findings of my research, the framework is empirically grounded yet anchored broadly in existing research. Accordingly, I argue that it is relevant for understanding ES design beyond my empirical case of DHIS2. The framework offers a novel conceptualization of ES design that tackles some of the challenges discussed in prior literature and adds to the discussions on how to best understand ES design (Bertram et al., 2012; Dittrich, 2014; Kallinikos, 2004a; Koch, 2007; Williams & Pollock, 2012).

To briefly reiterate the discussion in Chapters 2 and 3, to meaningfully understand ES design, it is argued that conceptualizations must capture ES as "extremely complex sociotechnical assemblages encompassing a huge variety of elements that are shaped over space and time" (Williams & Pollock, 2012, p. 14). Consequently, useful conceptualizations must help analyze "how they [ES] are inserted into organizational practices and also how they are evolving over time and across multiple sites of suppliers, users, and specialist intermediaries" (Williams & Pollock, 2012, p. 2). Whereas the concepts and findings offered by the stream of implementation studies are limited by a one-sided perspective favoring the user organizations' challenges and perspectives on implementation, the generification stream suffers from primarily focusing on the vendor side. The theoretical framework of the thesis brings the two perspectives together with the concepts of generic-level and implementation-level design, while emphasizing the software *design* part of implementation beyond the struggles of the adopting user organizations. The framework thereby shifts the focus to the overall dynamics of ES design while acknowledging ES implementation as an important part of ES design. It complements existing frameworks that attempt to describe and guide implementation in individual user organizations (e.g., Markus & Tanis, 2000) and that of generic-level design (e.g., generification; Pollock et al., 2007) and meta-design (Dittrich, 2014). Furthermore, it complements theories for studying ES as *biographies of artifacts* focusing on historical accounts of how the features of a given

ES have become what they are (Koch, 2007; Williams & Pollock, 2012), with a conceptualization that points more directly to how ES design can be strengthened and improved. Meanwhile, the framework integrates well with the existing conceptualization of ES ecosystems (Dittrich, 2014; Sarker et al., 2012; Wareham et al., 2014), highlighting not only the actors involved in design and their relations but also two key processes that the actors engage in and the dynamics between these, as well as positioning design infrastructure at the center of their efforts.

The framework portrays ES as a design infrastructure supporting implementation-level design. This poses an important pragmatic difference for research and practice. Seeing ES as a design infrastructure, rather than a *package*, a *solution*, or an *artifact* (Strong & Volkoff, 2010; Xu & Brinkkemper, 2007), immediately brings the focus toward how it can support processes of implementation-level design. The framework thus shifts the conversation to how diverse needs and contextual design can be supported rather than curbed and discouraged. It therefore presents a radically different perspective than what is offered by the implementation studies (Berente et al., 2016) and those concerned with the evil necessity of customization (Hustad et al., 2016; Rothenberger & Srite, 2009). This distinction in perspective is relevant for research and practice as ES vendors and partners increasingly seek for their ES solutions to become vehicles (or *design infrastructures*) supporting digitalization and digital innovation based on the unique needs of user organizations (Johnson, 2018). Portraying ES as a design infrastructure also brings attention to the relevance of perceiving generic-level design as a form of meta-design that aims to support further design. As such, the framework is aligned with earlier works on meta-design (Dittrich, 2014; Torkilsheyggi & Hertzum, 2017) but goes further by exploring the dynamics of such meta-design in greater detail in an ES context.

A key feature of the framework is the conceptualization of generic-level and implementation-level design as processes that respectively build resources for the design infrastructure and leverage these to design and innovate according to specific needs. Focusing on these processes, the thesis makes explicit an important dynamic (which remains somewhat vaguely defined in existing studies) that ES design at its core is about seeking to share design, development, and maintenance costs among adopting organizations. Accordingly, the shared processes of generic-level design should do the heavy work of development and maintenance, while processes of implementation-level design are ideally left with free or *affordable* design flexibility. Implementation-level design flexibility is thus not a mere function of whether it is *possible* to change the ES during implementation but whether it is possible *and* affordable, which relies on how much the solution can be changed without compromising the sharing of development and maintenance work. Making this dynamic explicit is valuable, as it allows studying the merit of different approaches to offering design flexibility.

The conceptualization of implementation-level design further rejects the separation between *design* and *implementation* made by several prior works (e.g., Williams & Pollock, 2012) and explicitly treats ES implementation as a context for professional software design and development. Furthermore, it does not tie generic-level or implementation-level design to specific actors. Consequently, the framework allows studying different ways of partitioning design and maintenance among multiple actors within an ES ecosystem. This ability is particularly useful when studying ES organized as extendible platforms, where the design and maintenance of generic software features, and thus generic-level design, are not solely driven by a single vendor.

In line with the pragmatist basis of the thesis, the theoretical framework adds to the toolbox of concepts and vocabulary that is useful for research and practice in coping with the real-world situation of designing ES. The framework's supporting roles in making sense of and addressing the real-world problem situation and in generating new, relevant, and more granular questions for further inquiry in the work of the design lab indicate its pragmatic relevance. For instance, highlighting the issue of affordable design flexibility and its connection to the partitioning of design and maintenance work between generic-level and implementation-level design offers an explanation that points to solutions and brings forth more granular problems. The analytical potential of the framework has thus far only been utilized to a modest extent, and I hope that it will be valuable for further research, not only internally in our ongoing work in the design lab, but also for researchers examining the design of other ES. For instance, one of the master's students in the design lab and I have used the framework to examine design and innovation practices in the SAP ecosystem, focusing on how the partners organize implementation-level design to be a fruitful arena for digital innovation in Norwegian organizations (Thomassen & Li, 2021). Thus far, the framework has been supportive in making sense of key dynamics in the SAP ecosystem and the partners' work with user organizations.

With the pragmatist basis of the thesis, its contribution can be further illustrated by the framework's ability to illuminate new questions for further scientific inquiry, which are relevant for ES practice. The framework highlights several interesting questions, such as when and how to take the design infrastructure route and how to offer affordable design flexibility. For the latter, it would be relevant to understand different vendors' strategies for offering affordable yet generative flexibility. Here, solutions surfacing in practitioner literature, such as *headless ES*, where the design of UIs is intentionally left to implementation-level design, and *low-code ES*, where implementations build a custom solution based on generic components, could be relevant (Zenner, 2020).

6.2 Contextual Implementation-Level Design

The theoretical framework and the findings point to the vital role played by processes of implementation-level design in designing ES to accommodate the specific needs of a diverse set of user organizations. As its second contribution, the thesis offers an extended appreciation of the nature of ES implementation as a context for design and concrete challenges of and conditions for contextual implementation-level design. The findings present implementation-level design as a dynamic, often long-term process. It involves a continuous interplay between an evolving set of generic software features and the specific and evolving needs of a user organization. In many cases, there is no clear post-implementation for the ES but an ongoing process of sociotechnical adjustments as new needs and digital possibilities emerge. The findings further highlight the important role of the partners as professional software designers in between a set of technical opportunities and organizational needs. With this, the thesis responds to calls for research on ES implementation as a context for *professional* software design and development (Dittrich, 2014; Dittrich & Vaucouleur, 2008; Sommerville, 2008). In doing so, the focus of the thesis differs from that of typical ES implementation studies, which tend to take the perspective of the individual user organization, with a primary "focus on organizational and social dynamics" (Berente et al., 2019, p. 26).

Whereas several ES implementation studies argue that ES imposes its own logic on the practices of user organizations and thus "embodies work procedures and practices that unfolded in an organisation favour certain ways of organizing" (Koch, 2007, p. 429; see also Davenport, 1998; Kallinikos, 2004a), the findings of this thesis show that this formative effect is also applicable to the processes of implementation-level design. The findings are in line with prior studies arguments that generic ES solutions "present a serious challenge for the design process, as they already provide a relatively comprehensive body of functionality that constrains the design space" (Pries-Heje & Dittrich, 2009, p. 52), which is also noted in earlier studies and echoed in later ones (Ellingsen & Hertzum, 2019; Martin et al., 2007; Mousavidin & Silva, 2017; Pollock & Cornford, 2002). The thesis delves deeper into this "serious challenge for the design process" (Pries-Heje & Dittrich, 2009, p. 52). The findings partly concur with prior arguments on the constraining effect of ES but provide a more accurate description: the generic software features and knowledge resources of the design infrastructure *guide and shape* the design process. They do so both directly by acting as lenses while carrying out design activities, such as requirement gathering, and indirectly by influencing the way that projects are configured and the implementation practices of the partners. As I argue in Paper 2, the design infrastructure "directs attention towards practices, needs, and challenges within the specific user organizations that can easily be addressed with generic features, and leaves other aspects in the dark […]. Where the

generic solution directs [its] focus seems to be manifested in the practices of the [partners], how projects are configured, and the focus of requirements gathering" (Li, 2021, p. 11).

The use of the term *formative effect* is a deliberate attempt to highlight an influence whose nature differs from that of what is labeled *constraining* in prior literature (Martin et al., 2007; Pries-Heje & Dittrich, 2009; Roland et al., 2017). In light of my findings, ES as a design infrastructure is not merely momentarily constraining in the sense that implementation-level designers want to achieve something and then learn that this is not possible. Rather, the design infrastructure shapes the practice and processes of implementation-level design, cultivating a necessary but potentially limiting mindset or lens that guides design during implementation. Nonetheless, as shown in the findings, the partners' practices have significant differences in contextual design, which means that it is possible to escape the formative effect with certain types of practices regarding the configuration and conduct of projects, that is, to escape the "mindless procedure" (Kallinikos, 2004a, p. 23) of standardized implementation processes promoted by the configurable generic software features and the knowledge resources. This requires an emphasis and a deliberate motivation for contextual design by the partners, which may be rewarded by securing innovative projects that artfully integrate specific practices with the right technology and identify areas for improvement based on generic features.

The thesis highlights several aspects of implementation-level design that raise new and interesting questions for further scientific inquiry with relevance for ES practice. Our research team are currently exploring some of these questions in the design lab, but I argue that they are also highly relevant to explore in other ES ecosystems. In general, the activity of negotiating implementation projects for contextual implementation-level design represents a valuable topic of further inquiry, for instance, to understand how partners balance between flexibility and predictability in projects. Many researchers also argue that contextual design is fruitful in supporting digital innovation as it helps make visible the challenges in the context of IT use, which can trigger new ways of organizing IT (e.g., Bygstad, 2010). Given that implementation-level design involves *two-sided monitoring* of technical possibilities and user needs, an interesting question is what key mechanisms or conditions are present for digital innovation to emerge during implementation-level design. The following are also relevant questions: What are the important elements of turning implementation-level design into a digitalization initiative rather than a digitization effort? How could projects be configured to best facilitate this? From the side of ES vendors, how can they cultivate a design infrastructure that promotes such practices?

6.3 Considerations for a Platform Strategy

The third contribution is primarily based on Paper 3 and relates to the merit and challenges of adopting a platform strategy to help design ES to accommodate the specific needs of a diverse set of user organizations. The contribution relates to two partly distinct yet (as this thesis shows) related conversations in IS literature – concerning ES platform strategies and ES design.

First, the thesis adds to the conversation on ES platform strategies (Foerderer et al., 2019; Huber et al., 2017; Kauschinger et al., 2021; Staub et al., 2021; Wareham et al., 2014), with insights into the challenges and considerations arising from the dynamics of ES design, in turn triggering issues that are unique compared with their much more discussed consumer software platform counterparts (as discussed in greater detail in Paper 3). Particularly, the challenges and considerations that I identify relate to issues regarding openness versus control (Wareham et al., 2014), means of governance (Huber et al., 2017), and challenges of *becoming an ES platform* (Bender, 2021).

Second, and most importantly, the thesis enriches the conversation on ES design (e.g., Dittrich, 2014; Mousavidin & Silva, 2017; Pollock et al., 2007) by offering concrete considerations for how to leverage a platform strategy to address the persistent challenge of designing ES to accommodate a diverse set of user organizations. The thesis goes beyond the important but basic realization that a platform architecture can help combine generic and specific software features (Roland et al., 2017), a trait it shares with a plethora of types of modular software architectures (Farhoomand, 2007). The findings show that the merit of any modularization attempting to provide affordable design flexibility to implementation-level design is based on how well it offers flexibility while retaining a sufficient amount of development and maintenance costs on processes of generic-level design. With this in mind, a platform architecture comprising core and apps is not, by any means, a silver bullet for balancing generic and specific needs. Nonetheless, in line with others, the findings show that a platform architecture can be a fruitful basis for offering flexibility for implementation-level design and for opening up generic-level design to partners (Roland et al., 2017).

While the thesis shows that a platform strategy can be an integral part of organizing a design infrastructure and a way of operationalizing meta-design (Dittrich, 2014), I stress that my concept of design infrastructure is not equated with an *ES platform*. A platform strategy is but one way of organizing ES design infrastructures. There are many alternative ways of organizing to support implementation-level design (Bertram et al., 2012; Mousavidin & Silva, 2017), in terms of how to offer implementation-level design flexibility (e.g., configurability, componentization, extensibility, low code, and headless), as well as how to partition design and maintenance work among actors (e.g., a single vendor in charge of both generic-level and implementation-level design, an

ecosystem including partners specializing in implementation, and platform ecosystems, supporting partners in offering generic apps).

The thesis points at some interesting phenomena for further scientific inquiry, which would be relevant for the practice of ES design and ES platform strategies. An interesting finding, which is not fully unpacked in this thesis, is that partner-driven generic-level design often originates from implementation-level design. It would be relevant to study when, why, and how partners make the move from designing a custom app during implementation-level design to offering it to the wider ecosystem through generic-level design. Furthermore, it is pertinent to understand how the ES vendor or the platform owner can accommodate, encourage, and incentivize more implementation-specific innovations to be generified, ensuring that these are made available in the design infrastructure when they have generic relevance. Concretely related to the considerations identified in Paper 3, it is also valuable to explore different modes for design and maintenance partitioning, for addressing uncertainties regarding the dependence on generic software features designed and maintained by partners, and for ensuring design infrastructure cognizance and navigability.

6.4 Methodological Contributions

For RQ 2, which is addressed in the Research Approach chapter, I offer a methodological contribution. The twofold contribution is based on the design lab as a research approach for the thesis research project and the approach to engaged scholarship presented in Paper 4.

First, closely related to the contributions for RQ 1 and the theoretical framework in particular, the thesis offers a conceptualization of the *design lab* as an approach to studying ES design in collaboration with relevant practitioners. Studying ES design is discussed as a major challenge in existing literature, being subject to "shaping […] distributed in time and space" (Williams & Pollock, 2012, p. 3), potentially by an ecosystem of multiple actors operating in different constituencies (Dittrich, 2014; Koch, 2007). Furthermore, the majority of existing studies are limited by focusing on single locales, primarily the level of implementation. Some scholars (e.g., Koch, 2007; Williams & Pollock, 2012) suggest tackling these limitations by studying ES and its surrounding ecosystems as biographies, focusing on longitudinal research on how and why the software features come to be as they are. Others argue for studying ES by examining the ecosystem of actors around it (Dittrich, 2014; Sarker et al., 2012). Most closely aligned with the latter, the thesis shows that a way to organize the study of ES design in a manner geared toward understanding how to strengthen design is by seeing ES as a design infrastructure supporting implementation-level design. A design lab can focus on understanding and improving processes of generic-level design, the design infrastructure, and implementation-level design (illustrated in Figure 4.8, Chapter 4).

Accordingly, the design lab as an approach allows studying ES design by systemically focusing on multiple locales and resources from the perspective of multiple relevant actors. This is done while maintaining a focus on the key mission of ES to offer relevant yet cost-effective software solutions to a diverse set of user organizations. The thesis shows how engaged scholarship can be used as a methodological basis for such a design lab, allowing the exploration of challenges and solutions by combining diagnostic, design, and intervention-oriented inquiries into processes and products of ES design.

Although the design lab has emerged in the rather unique context of DHIS2 and the closely related research group at the UiO, the approach of the design lab and the design lab as a conceptual and methodological package could serve as an inspiration for studying and strengthening design in other ES ecosystems.

Second, the approach to engaged scholarship developed as part of the thesis project, which is presented in Paper 4, is relevant to IS researchers concerned with engaged scholarship more generally. Existing literature offers guidance for planning and carrying out specific forms of engaged scholarship, including CS, AR, DSR, and ADR, and some studies offer reflections on the positive merit of potentially combining them (Davison et al., 2021; Goldkuhl, 2011; Mathiassen, 2002; Nielsen, 2020). However, there is limited support for entering and organizing projects so that the form(s) of inquiry can be selected and potentially combined as the understanding of the problem and the project evolve.

The approach described in the thesis may prove useful for other researchers who find themselves in the situation of planning engaged research projects where any informed selection of a specific form of engaged inquiry is difficult due to limited knowledge of the problem situation and the researcher–practitioner collaboration. The model presented in Paper 4 suggests using CS as the initial form of inquiry to develop a greater appreciation of the real-world problem situation and the possibilities that lie in the researcher–practitioner collaboration. Other forms of inquiry are potentially selected when they emerge as relevant and feasible. As suggested to be relevant by others (Davison et al., 2021; Nielsen & Persson, 2016), the model also affords and encourages combining several forms of inquiry, either in sequence (e.g., CS followed by AR) or in parallel. With this, the model has proven to scale well in the research of the design lab, serving as the basis for both the overall research project and the individual research projects of the master's students. For the overall project, a shared problem formulation helps bind the different inquiries together in sequence and in parallel, while the individual research projects dominantly comprise CS, potentially followed by AR, DSR, or ADR if relevant and feasible.

Paper 4 mostly elaborates on the issue of selecting form(s) of engaged inquiry in research projects based on relevance and feasibility. Several interesting aspects, which

are only briefly examined in this thesis and in Paper 4, represent avenues for further studies on engaged scholarship in IS research. Particularly, unpacking the activities of problem formulation and researcher–practitioner negotiation to a greater extent would be valuable as these are both challenging and very formative of the research project. For the problem formulation, a relevant challenge for further inquiry is how to best organize problem formulations that are shared by several researchers and research initiatives within larger research projects or programs. How and when to involve practitioners in problem formulation could also be paid more attention, as argued by Nielsen and Persson (2016), for instance, based on the knowledge of problem formulation from the soft systems methodology (Checkland & Poulter, 2006). Unpacking the role and nature of abductive leaps and casing in problem formulation (e.g., when and how) is also relevant as it plays an essential role in connecting the real-world problem situation to a pertinent body of knowledge and conceptual scheme(s) in academic literature.

6.5 Implications for Practice

Having discussed how the thesis contributes to IS research, I now outline a few practical implications of the findings for ES vendors, partners specializing in implementation, and user organizations.

6.5.1 For ES vendors

The format of the answer to RQ 1 is in itself directed toward the practice of ES vendors. To summarize a bit more broadly, my findings indicate (in line with existing literature) that generic-level design cannot fully accommodate a diverse set of user organizations exclusively in the form of ready-to-use software features. Accordingly, a key question for vendors is what role they have to play: a provider of a ready-made software solution with the potential challenges this introduces during implementation, or a provider of a design infrastructure for implementation-level design, which comes with its own challenges and concerns. Following the latter route, another question is how to partition design and maintenance work between the processes of generic-level and implementation-level design, as well as in terms of actors. A vendor may control both processes, partners may be engaged in implementation-level design, or a platform strategy may be adopted to engage partners also in generic-level design.

If aiming to offer ES as a design infrastructure, the vendor must offer *affordable* design flexibility, partitioning for development, and maintenance costs to remain on the processes of generic-level design, while providing relevant flexibility for implementation-level design. In cultivating a design infrastructure, an important consideration for vendors is how the infrastructure shapes practices, projects, and processes in certain directions. Vendors must thus think strategically and holistically

about what kinds of design and innovation they seek to promote during implementation when cultivating the design infrastructure.

6.5.2 For partners

The thesis shows that with extended design flexibility for implementation-level design, the partners' role becomes increasingly important. The findings indicate that the partners who are successful in carrying out contextual design and innovation in implementation projects tend to be so because of their focus on configuring the right project scope and structure, while being active in utilizing the possibilities of the design infrastructure. The partners should therefore take an active role in configuring projects with a focus on negotiating for contextual design, and organizing the project around high-level goals rather than specific technical requirements. Furthermore, a virtue of the successful partners is their cognizance of the possibilities afforded by the design infrastructure while continuously monitoring the evolving practices and needs of the user organizations. The partners should take this important role seriously and attempt to establish long-term relationships with user organizations.

6.5.3 For user organizations

Finally, for the user organizations, it is important to realize that ES implementation is much more than merely inserting a standard software package in the organization. Rather, it should be recognized as a full-fledged software design and innovation project if seeking to maximize the potential for using implementation as a driver of *techno-change* (Markus, 2004). Accordingly, awareness of the need for and the benefit of budgeting for contextual design activities and the potential benefits of goal-oriented rather than requirement-oriented project configurations are important. During the selection and procurement of partners and generic ES solutions, user organizations should be aware that they commit to not only a static package but also an evolving design infrastructure and its surrounding ecosystem of the vendor(s) and partners. In selecting a partner, user organizations should seek to establish long-term relationships with someone that takes seriously the virtue of continuously monitoring and matching the technological possibilities of the design infrastructure and their evolving needs.

6.6 Limitations

I end this chapter by discussing some of the limitations of the thesis. As I argue from existing perspectives, the one presented in this thesis inevitably suffers from some limitations by favoring certain actors, processes, and resources at the expense of others.

While the thesis has the strength of examining a case of ES design from the perspective of multiple actors and processes of design, my findings are based on a single case, and one with some unique attributes. The unique history of DHIS2 as based on a long-term research project, being open source, and primarily targeting low-income and middle-

income countries may imply several different dynamics from other ES related to governance, funding, and incentives. I have attempted to tackle this by emphasizing the features that (based on related academic literature) are shared with other ES and particularly pertinent to the aim of designing and maintaining a set of software features for use by several organizations. Particularly, the theoretical framework, although emerging from the empirical case, is developed based on existing literature, and I thus argue for its more general relevance. However, inquiries into other ES may find that the dynamics between generic-level and implementation-level design, the conditions for contextual implementation-level design, and key challenges of and considerations for an ES platform strategy differ due to various modes of governance, differences in how design infrastructures are organized, and diverse capacities and motivations for the partners, the user organizations, and the vendor.

For instance, the issues related to the dependence on generic apps offered by partners during implementation-level design could be expected to differ based on governance models and the financial incentives that the partners see in sustaining generic apps. Nonetheless, uncertainties tied to dependability will most probably represent a relevant challenge. The setup of the design lab with its rather unique access to the DHIS2 vendor and the partners based on the well-established HISP research program may also be difficult to export to other contexts. I thus stress the contribution of the design lab as a conceptual and methodological package to study ES design as an *inspiration* to research on other ES.

The thesis has primarily focused on the processes of generic-level and implementation-level design, as well as the design infrastructure between them. It could be relevant to study the nature of design infrastructure in greater detail, possibly leveraging existing concepts in infrastructure literature to a greater extent. An example is how it is perceived and leveraged similarly or differently during implementation-level design.

Aside from the insider CS work in India, much of the inquiries into implementation-level design are based on outsider CS and thus, the reported experiences by participants. There may be a greater potential for theorizing implementation-level design based on more in-depth engagement in such processes. The initial plan when visiting the partner in Mozambique was to establish a collaboration where several master's students and I could actively participate in some of their ongoing projects. The idea was to adopt a form of cooperative method development (Dittrich et al., 2008), a specific form of DSR/ADR emphasizing collaborative strengthening of software methods with practitioners. Due to the COVID-19 pandemic, which halted a planned collaboration with Mozambique, the proceeding inquiries involved a broader examination of practices and less insider examination of implementation-level design, shifting the focus somewhat to the relation between partners and vendors. Accordingly, the thesis predominantly focuses on the dynamics between generic-level and implementation-

level design and less on the important relationship between the partner and the user organization during implementation-level design. As detailed in Section 6.2, there remains a huge potential for exploring several aspects of implementation-level design in more detail.

References

á Torkilsheyggi, A., & Hertzum, M. (2017). Incomplete by Design. A study of a design-in-use approach to systems implementation. *Scandinavian Journal of Information Systems*, *29*(2), 2.

Adu-Gyamfi, E., Nielsen, P., & Sæbø, J. I. (2019). The dynamics of a global health information systems research and implementation project. *SHI 2019. Proceedings of the 17th Scandinavian Conference on Health Informatics, November 12-13, 2019, Oslo, Norway*, *161*, 73–79.

Ågerfalk, P. J. (2010). Getting pragmatic. *European Journal of Information Systems*, *19*, 251–256.

Alavi, H. S., Lalanne, D., & Rogers, Y. (2020). The five strands of living lab. *ACM Transactions on Computer-Human Interaction*, *27*(2). Scopus. https://doi.org/10.1145/3380958

Antero, M. C., & Bjørn-Andersen, N. (2013). Why a partner ecosystem results in superior value: A comparative analysis of the business models of two ERP vendors. *Information Resources Management Journal (IRMJ)*, *26*(1), 12–24.

Bacon, M. (2012). *Pragmatism: An introduction*. Polity.

Baxter, G., & Sommerville, I. (2011). Socio-technical systems: From design methods to systems engineering. *Interacting with Computers*, *23*(1), 4–17.

Bechara, J., & Van de Ven, A. (2007). Philosophy of science underlying engaged scholarship. In *Engaged scholarship: A guide for organizational and social research*. Oxford University Press.

Bender, B. (2021). Effects of Platform Coring on Complements Usage—The Salesforce Case. In *Platform Coring on Digital Software Platforms* (pp. 201–219). Springer.

Berente, N., Lyytinen, K., Yoo, Y., & King, J. L. (2016). Routines as shock absorbers during organizational transformation: Integration, control, and NASA's enterprise information system. *Organization Science*, *27*(3), 551–572.

Berente, N., Lyytinen, K., Yoo, Y., & Maurer, C. (2019). Institutional logics and pluralistic responses to enterprise system implementation: A qualitative meta-analysis. *MIS Quarterly*, *43*(3).

Berente, N., & Yoo, Y. (2012). Institutional contradictions and loose coupling: Postimplementation of NASA's enterprise information system. *Information Systems Research*, *23*(2), 376–396.

Bergvall-Kareborn, B., & Stahlbrost, A. (2009). Living Lab: An open and citizen-centric approach for innovation. *International Journal of Innovation and Regional Development*, *1*(4), 356–370.

Bertram, M., Schaarschmidt, M., & von Kortzfleisch, H. F. (2012). Customization of Product Software: Insight from an Extensive IS Literature Review. In *Shaping the Future of ICT Research. Methods and Approaches* (pp. 222–236). Springer.

Binder, T., & Brandt, E. (2008). The Design: Lab as platform in participatory design research. *Co-Design*, *4*(2), 115–129.

Binder, T., Brandt, E., Halse, J., Foverskov, M., Olander, S., & Yndigegn, S. (2011). Living the (co-design) Lab. *Nordes*, *4*.

Blomberg, J., Suchman, L., & Trigg, R. H. (1996). Reflections on a work-oriented design project. *Human–Computer Interaction*, *11*(3), 237–265.

Bødker, S., & Buur, J. (2002). *The design collaboratorium: A place for usability design*. Association for Computing Machinery. https://doi.org/10.1145/513665.513670

Braa, J., Monteiro, E., & Sahay, S. (2004). Networks of action: Sustainable health information systems across developing countries. *Mis Quarterly*, 337–362.

Brehm, L., Heinzl, A., & Markus, M. L. (2001). Tailoring ERP systems: A spectrum of choices and their implications. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, 9 pp.

Bygstad, B. (2010). Generative mechanisms for innovation in information infrastructures. *Information and Organization*, *20*(3–4), 156–168.

Checkland, P., & Poulter, J. (2006). Soft systems methodology. In *Systems approaches to making change: A practical guide* (pp. 201–253). Springer.

Ciborra, C. (2000). *From control to drift: The dynamics of corporate information infastructures*. Oxford University Press on Demand.

Conboy, K., Fitzgerald, G., & Mathiassen, L. (2012). Qualitative methods research in information systems: Motivations, themes, and contributions. *European Journal of Information Systems*, *21*(2), 113–118.

Cronen, V. E. (2001). Practical theory, practical art, and the pragmatic-systemic account of inquiry. *Communication Theory*, *11*(1), 14–35.

Damsgaard, J., & Karlsbjerg, J. (2010). Seven principles for selecting software packages. *Communications of the ACM*, *53*(8), 63–71.

Davenport, T. H. (1998). Putting the enterprise into the enterprise system. *Harvard Business Review*, *76*(4).

Davison, R. M., Martinsons, M. G., & Malaurent, J. (2021). Research Perspectives: Improving Action Research by Integrating Methods. *Journal of the Association for Information Systems*, *22*(3), 1.

de Reuver, M., Sørensen, C., & Basole, R. C. (2018). The digital platform: A research agenda. *Journal of Information Technology*, *33*(2), 124–135.

Dewey, J. (1931). The Development of American Pragmatism." Philosophy and Civilization. *GP Putnam's Sons, New York*.

Dewey, J. (1969a). *The collected works of John Dewey 1882-1953: Vol. Later Works 4*. Southern Illinois University Press.

Dewey, J. (1969b). *The collected works of John Dewey 1882-1953: Vol. Middle Works 12*. Southern Illinois University Press.

Dittrich, Y. (2014). Software engineering beyond the project–Sustaining software ecosystems. *Information and Software Technology*, *56*(11), 1436–1456.

Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., & Lindeberg, O. (2008). Cooperative method development. *Empirical Software Engineering*, *13*(3), 231–260.

Dorst, K. (2011). The core of 'design thinking'and its application. *Design Studies*, *32*(6), 521–532.

Draxler, S., & Stevens, G. (2011). Supporting the collaborative appropriation of an open software ecosystem. *Computer Supported Cooperative Work (CSCW)*, *20*(4–5), 403–448.

Ellingsen, G., & Hertzum, M. (2019). User participation in the implementation of large-scale suite systems in healthcare. *Proceedings of the 7th International Conference on Infrastructures in Healthcare*, *4*(3). https://doi.org/10.18420/ihc2019_002

Elragal, A., Haddara, M., & Hustad, E. (2020). Rejuvenating Enterprise Systems. *Scandinavian Journal of Information Systems*, *32*(2), 127–138.

Farhoomand, A. (2007). Opening up of the software industry: The case of SAP. *Eighth World Congress on the Management of EBusiness (WCMeB 2007)*, 8–8.

Fischer, G., & Giaccardi, E. (2006). Meta-design: A framework for the future of end-user development. In *End user development* (pp. 427–457). Springer.

Foerderer, J., Kude, T., Schuetz, S. W., & Heinzl, A. (2019). Knowledge boundaries in enterprise software platform development: Antecedents and consequences for platform governance. *Information Systems Journal*, *29*(1), 119–144.

Gizaw, A. A., Bygstad, B., & Nielsen, P. (2017). Open generification. *Information Systems Journal*, *27*(6), 619–642.

Goldkuhl, G. (2011). The research practice of practice research: Theorizing and situational inquiry. *Systems, Signs & Actions*, *5*(1), 7–29.

Goldkuhl, G. (2012). Pragmatism vs interpretivism in qualitative information systems research. *European Journal of Information Systems*, *21*(2), 135–146.

Goldkuhl, G. (2020). Design Science Epistemology: A Pragmatist Inquiry. *Scandinavian Journal of Information Systems*, *32*(1), 39–80.

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, Å. (2003). Key principles for user-centred systems design. *Behaviour and Information Technology*, *22*(6), 397–409.

Haack, S. (2000). *Manifesto of a passionate moderate: Unfashionable essays*. University of Chicago Press.

Haines, M. N. (2009). Understanding enterprise system customization: An exploration of implementation realities and the key influence factors. *Information Systems Management*, *26*(2), 182–198.

Halckenhäußer, A., Förderer, J., & Heinzl, A. (2020). *Platform governance mechanisms: An integrated literature review and research directions*. European Conference on Information Systems.

Hanseth, O., & Lyytinen, K. (2010). Design theory for dynamic complexity in information infrastructures: The case of building internet. *Journal of Information Technology*, *25*(1), 1–19.

Hanseth, O., Monteiro, E., & Hatling, M. (1996). Developing information infrastructure: The tension between standardization and flexibility. *Science, Technology, & Human Values*, *21*(4), 407–426.

Hocko, J. (2011). User-centered design in procured software implementations. *Journal of Usability Studies*, *6*(2), 60–74.

Huber, T. L., Kude, T., & Dibbern, J. (2017). Governance practices in platform ecosystems: Navigating tensions between cocreated value and governance costs. *Information Systems Research*, *28*(3), 563–584.

Hustad, E., Haddara, M., & Kalvenes, B. (2016). ERP and organizational misfits: An ERP customization journey. *Procedia Computer Science*, *100*, 429–439.

Jæger, B., Bruckenberger, S. A., & Mishra, A. (2020). Critical Success Factors for ERP Consultancies. *Scandinavian Journal of Information Systems*, *32*(2), 169–202.

Johnson, A. (2018). *Design Thinking with SAP*. SAP PRESS.

Jones, D., & Gregor, S. (2007). The anatomy of a design theory. *Journal of the Association for Information Systems*, *8*(5), 1.

Kaipio, J., Lääveri, T., Hyppönen, H., Vainiomäki, S., Reponen, J., Kushniruk, A., Borycki, E., & Vänskä, J. (2017). Usability problems do not heal by themselves: National survey on

physicians' experiences with EHRs in Finland. *International Journal of Medical Informatics*, *97*, 266–281.

Kallinikos, J. (2004a). Deconstructing information packages: Organizational and behavioural implications of ERP systems. *Information Technology & People*, *17*(1), 8–30.

Kallinikos, J. (2004b). Farewell to constructivism: Technology and context-embedded action. *The Social Study of Information and Communication Technology: Innovation, Actors, and Contexts*, *140*, 161.

Kallinikos, J. (2009). Book Review: Neil Pollock & Robin Williams Software and Organisations: The biography of the enterprise-wide system or how SAP conquered the world: Routledge: Abingdon. *Organization Studies*, *30*(8), 912–916.

Kauschinger, M., Schreieck, M., Böhm, M., & Krcmar, H. (2021). Knowledge Sharing in Digital Platform Ecosystems–A Textual Analysis of SAP's Developer Community. *International Conference on Wirtschaftsinformatik*.

Kharabe, A., & Lyytinen, K. J. (2012). *Is implementing ERP like pouring concrete into a company? Impact of enterprise systems on organizational agility*. *33*.

Khoo, H. M., Robey, D., & Rao, S. V. (2011). An exploratory study of the impacts of upgrading packaged software: A stakeholder perspective. *Journal of Information Technology*, *26*(3), 153–169.

Koch, C. (2007). ERP-a moving target. *International Journal of Business Information Systems*, *2*(4), 426.

Leidner, D. E. (2018). Review and theory symbiosis: An introspective retrospective. *Journal of the Association for Information Systems*, *19*(6), 1.

Li, M. (2021). *Generic Enterprise Software implementation as Context for User-Oriented Design: Three Conditions and their Implications for Vendors.* 12th Scandinavian Conference on Information Systems (SCIS), Orkanger, Norway.

Light, B. (2001). The maintenance implications of the customization of ERP software. *Journal of Software Maintenance and Evolution: Research and Practice*, *13*(6), 415–429.

Light, B., & Sawyer, S. (2007). Locating packaged software in information systems research. *European Journal of Information Systems*, *16*(5), 527–530.

Lorino, P. (2018). *Pragmatism and organization studies*. Oxford University Press.

Magnusson, J., Klingberg, J., Enquist, H., Oskarsson, B., Nilsson, A., & Gidlund, A. (2010). All Aboard: ERP Implementation as Participatory Design. *American Conference on Information Systems*, 253.

Magnusson, J., & Nilsson, A. (2013). Introducing app stores into a packaged software ecosystem: A negotiated order perspective. *International Journal of Business Information Systems*, *14*(2), 223–237.

Markus, M. L. (2004). Technochange management: Using IT to drive organizational change. *Journal of Information Technology*, *19*(1), 4–20.

Markus, M. L., & Tanis, C. (2000). The enterprise systems experience-from adoption to success. *Framing the Domains of IT Research: Glimpsing the Future through the Past*, *173*(2000), 207–173.

Martin, D., Mariani, J., & Rouncefield, M. (2007). Managing integration work in an NHS electronic patient record (EPR) project. *Health Informatics Journal*, *13*(1), 47–56.

Mathiassen, L. (2002). Collaborative practice research. *Information Technology & People*.

Mathiassen, L. (2017). Designing engaged scholarship: From real-world problems to research publications. *Engaged Management Review*, *1*(1), 2.

Mathiassen, L., & Nielsen, P. A. (2008). Engaged scholarship in IS research. *Scandinavian Journal of Information Systems*, *20*(2), 1.

Monteiro, E., Pollock, N., Hanseth, O., & Williams, R. (2013). From artefacts to infrastructures. *Computer Supported Cooperative Work (CSCW)*, *22*(4–6), 575–607.

Morgan, D. L. (2014). Pragmatism as a paradigm for social research. *Qualitative Inquiry*, *20*(8), 1045–1053.

Mousavidin, E., & Silva, L. (2017). Theorizing the configuration of modifiable off-the-shelf software. *Information Technology & People*, *30*(4), 887–909.

Mozaffar, H., Williams, R., Cresswell, K., & Sheikh, A. (2018). Anglicization of hospital information systems: Managing diversity alongside particularity. *International Journal of Medical Informatics*, *119*, 88–93.

Nicholson, B., Nielsen, P., Saebo, J., & Sahay, S. (2019). Exploring Tensions of Global Public Good Platforms for Development: The Case of DHIS2. In P. Nielsen & H. C. Kimaro (Eds.), *Information and Communication Technologies for Development. Strengthening Southern-Driven Cooperation as a Catalyst for ICT4D* (pp. 207–217). Springer International Publishing. https://doi.org/10.1007/978-3-030-18400-1_17

Nielsen, J. (1994). Usability laboratories. *Behaviour & Information Technology*, *13*(1–2), 3–8.

Nielsen, P. A. (2020). Problematizing in IS Design Research. *International Conference on Design Science Research in Information Systems and Technology*, 259–271.

Nielsen, P. A., & Persson, J. S. (2016). Engaged problem formulation in IS research. *Communications of the Association for Information Systems*, *38*(1), 35.

Norman, D. A. (2013). *The design of everyday things* (Revised and expanded edition). Basic Books. https://mitpress.mit.edu/books/design-everyday-things-revised-and-expanded-edition

Parthasarathy, S., & Sharma, S. (2016). Efficiency analysis of ERP packages—A customization perspective. *Computers in Industry*, *82*, 19–27.

Pipek, V., & Wulf, V. (2009). Infrastructuring: Toward an integrated perspective on the design and use of information technology. *Journal of the Association for Information Systems*, *10*(5), 1.

Pollock, N., & Cornford, J. (2002). Fitting standard software to non-standard organisations. *Proceedings of the 2002 ACM Symposium on Applied Computing*, 721–725.

Pollock, N., & Hyysalo, S. (2014). The Business of Being a User. *MIS Quarterly*, *38*(2), 473–496.

Pollock, N., Williams, R., & D'Adderio, L. (2007). Global software and its provenance: Generification work in the production of organizational software packages. *Social Studies of Science*, *37*(2), 254–280.

Pries-Heje, L., & Dittrich, Y. (2009). ERP implementation as design: Looking at participatory design for means to facilitate knowledge integration. *Scandinavian Journal of Information Systems*, *21*(2), 4.

Roland, L. K., Sanner, T. A., Sæbø, J. I., & Monteiro, E. (2017). P for Platform: Architectures of large-scale participatory design. *Scandinavian Journal of Information Systems*, *29*(2).

Rolland, K. H., & Monteiro, E. (2002). Balancing the local and the global in infrastructural information systems. *The Information Society*, *18*(2), 87–100.

Rothenberger, M. A., & Srite, M. (2009). An investigation of customization in ERP system implementations. *IEEE Transactions on Engineering Management*, *56*(4), 663–676.

Sahay, S., Nielsen, P., Faujdar, D., Kumar, R., & Mukherjee, A. (2018). Frugal Digital Innovation and Living Labs: A Case Study of Innovation in Public Health in India. *International Conference on Information Systems*. https://aisel.aisnet.org/icis2018/innovation/Presentations/10

Sarker, S., Sarker, S., Sahaym, A., & Bjørn-Andersen, N. (2012). Exploring value cocreation in relationships between an ERP vendor and its partners: A revelatory case study. *MIS Quarterly*, 317–338.

Seddon, P. B., Calvert, C., & Yang, S. (2010). A multi-project model of key factors affecting organizational benefits from enterprise systems. *MIS Quarterly*, 305–328.

Sein, M., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *Mis Quarterly*.

Sestoft, P., & Vaucouleur, S. (2008). Technologies for evolvable software products: The conflict between customizations and evolution. In *Advances in Software Engineering* (pp. 216–253). Springer.

Sia, S. K., & Soh, C. (2007). An assessment of package–organisation misalignment: Institutional and ontological structures. *European Journal of Information Systems*, *16*(5), 568–583.

Simon, H. A. (1996). *The sciences of the artificial*. MIT press.

Simonsen, J. (2009). A Concern for Engaged Scholarship: The challenges for action research projects. *Scandinavian Journal of Information Systems*, *21*(2), 1.

Singh, C., & Pekkola, S. (2021). Packaged Enterprise System Customization–A Systematic Literature Review. *Proceedings of the 54th Hawaii International Conference on System Sciences*, 6743.

Soh, C., Kien, S. S., & Tay-Yap, J. (2000). Cultural fits and misfits: Is ERP a universal solution? *Communications of the ACM*, *43*(4), 47–47.

Soh, C., & Sia, S. K. (2008). The challenges of implementing" vanilla" versions of enterprise systems. *MIS Quarterly Executive*, *4*(3), 6.

Sommerville, I. (2008). Construction by configuration: Challenges for software engineering research and practice. *19th Australian Conference on Software Engineering (ASWEC 2008)*, 3–12.

Star, S. L., & Ruhleder, K. (1996). Steps toward an ecology of infrastructure: Design and access for large information spaces. *Information Systems Research*, *7*(1), 111–134.

Staub, N., Haki, K., Aier, S., Winter, R., & Magan, A. (2021, June 15). *Evolution of B2B Platform Ecosystems: What Can Be Learned from Salesforce?* European Conference on Information Systems.

Stevens, G., Pipek, V., & Wulf, V. (2009). Appropriation Infrastructure: Supporting the Design of Usages. In V. Pipek, M. B. Rosson, B. de Ruyter, & V. Wulf (Eds.), *End-User Development* (Vol. 5435, pp. 50–69). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00427-8_4

Strong, D. M., & Volkoff, O. (2010). Understanding Organization—Enterprise system fit: A path to theorizing the information technology artifact. *MIS Quarterly*, 731–756.

Suchman, L. (1993). Working relations of technology production and use. *Computer Supported Cooperative Work*, *2*(1–2), 21–39.

Suchman, L. (2002). Located accountabilities in technology production. *Scandinavian Journal of Information Systems*, *14*(2), 7.

Swan, J., Newell, S., & Robertson, M. (1999). The illusion of 'best practice'in information systems for operations management. *European Journal of Information Systems*, *8*(4), 284–293.

Sykes, T. A., & Venkatesh, V. (2017). Explaining post-implementation employee system use and job performance: Impacts of the content and source of social network ties. *MIS Quarterly*, *41*(3), 917–936.

Sykes, T. A., Venkatesh, V., & Johnson, J. L. (2014). Enterprise system implementation and employee job performance: Understanding the role of advice networks. *MIS Quarterly*, *38*(1), 51–72.

Tan, B., Pan, S. L., Chen, W., & Huang, L. (2020). Organizational Sensemaking in ERP Implementation: The Influence of Sensemaking Structure. *MIS Quarterly*, *44*(4).

Tavory, I., & Timmermans, S. (2014). *Abductive analysis: Theorizing qualitative research*. University of Chicago Press.

Thomassen, M. L., & Li, M. (2021). *Enterprise Software Implementation as Context for Digital Innovation*. Selected Papers of the IRIS.

Timmermans, S., & Tavory, I. (2012). Theory construction in qualitative research: From grounded theory to abductive analysis. *Sociological Theory*, *30*(3), 167–186.

Tiwana, A. (2013). *Platform ecosystems: Aligning architecture, governance, and strategy*. Newnes.

Topi, H., Lucas, W. T., & Babaian, T. (2005). *Identifying Usability Issues with an ERP Implementation*. 128–133.

van Beijsterveld, J. A., & Van Groenendaal, W. J. (2016). Solving misfits in ERP implementations by SMEs. *Information Systems Journal*, *26*(4), 369–393.

Van de Ven, A. H. (2007). *Engaged Scholarship: A Guide for Organizational and Social Research*. OUP Oxford.

Van den Hooff, B., & Hafkamp, L. (2017). *Dealing with dissonance: Misfits between an EHR system and medical work practices*. International Conference on Information Systems.

Van Fenema, P. C., Koppius, O. R., & Van Baalen, P. J. (2007). Implementing packaged enterprise software in multi-site firms: Intensification of organizing and learning. *European Journal of Information Systems*, *16*(5), 584–598.

van Groenendaal, W., & van der Hoeven, H. (2008). Best Practices in ERP: How good are they? *Proceedings of the 3rd SIKS/BENAIS Conference on Enterprise Information Systems*, 1–13.

Vidgen, R., & Braa, K. (1997). Balancing interpretation and intervention in information system research: The action case approach. In *Information systems and qualitative research* (pp. 524–541). Springer.

Vilpola, I. H. (2008). A method for improving ERP implementation success by the principles and process of user-centred design. *Enterprise Information Systems*, *2*(1), 47–76.

Wagner, E. L., Scott, S. V., & Galliers, R. D. (2006). The creation of 'best practice'software: Myth, reality and ethics. *Information and Organization*, *16*(3), 251–275.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, *15*(3), 320–330.

Wareham, J., Fox, P. B., & Cano Giner, J. L. (2014). Technology ecosystem governance. *Organization Science*, *25*(4), 1195–1215.

Williams, A. (2009). User-centered Design, Activity-centered Design, and Goal-directed Design: A Review of Three Methods for Designing Web Applications. *Proceedings of the 27th ACM*

International Conference on Design of Communication, 1–8. https://doi.org/10.1145/1621995.1621997

Williams, R., & Pollock, N. (2012). Research commentary—Moving beyond the single site implementation study: How (and why) we should study the biography of packaged enterprise solutions. *Information Systems Research*, *23*(1), 1–22.

Wong, W.-P., Veneziano, V., & Mahmud, I. (2016). Usability of Enterprise Resource Planning software systems: An evaluative analysis of the use of SAP in the textile industry in Bangladesh. *Information Development*, *32*(4), 1027–1041.

Xu, L., & Brinkkemper, S. (2007). Concepts of product software. *European Journal of Information Systems*, *16*(5), 531–541.

Zahlsen, Ø. K., Svanæs, D., Faxvaag, A., & Dahl, Y. (2020). Understanding the Impact of Boundary Conditions on Participatory Activities. *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, 1–11.

Zenner, R. (2020). *Benefits of Headless Commerce—How businesses can drive innovation and increase agility by decoupling their applications*. Commercetools.com. www.commercetools.com

Appendices: The Papers

I

MAKING USABLE GENERIC SOFTWARE. A MATTER OF GLOBAL OR LOCAL DESIGN?

Research paper

Li, Magnus, University of Oslo, Oslo, Norway, magl@ifi.uio.no
Nielsen, Petter, University of Oslo, Oslo, Norway, pnielsen@ifi.uio.no

Abstract

Usability is widely acknowledged as a desirable trait of software, referring to how usable it is to a specific set of users. However, when software is developed as generic packages, aimed at supporting variety, designing user interfaces with sufficient sensitivity to use-contexts is a challenge. Extant literature has documented this challenge and established that solving usability-related problems are difficult, both during software development and implementation. Adding to this discussion, this paper contributes by developing a framework to analyze what characterizes usability-related design of generic software. This includes two levels of design; generic-level and implementation-level, and two types of design; design for use and design for design. We apply this conceptual framework on an empirical case based on an ongoing action research project where a global generic health software is implemented in a large state in India. From the analysis we argue that attempts to strengthen usability of generic software require a holistic intervention, considering design on both 'global' and 'local' level. Of particular importance is how usable the generic software and other design-resources are when implementers are customizing the software. We coin this aspect of design as meta-usability, which represent what we see as an avenue for further research.

Keywords: Usability, Generic Software, Implementation-level design, Meta-usability.

1 Introduction

A substantial portion of the software implemented in organizations today are 'generic' or 'off-the-shelf' type of software, developed to work across an array of organizational settings and use-cases (Baxter & Sommerville, 2011). Typical examples are Enterprise resource planning software (ERPs) (Dittrich, 2014; Dittrich, Vaucouleur, & Giff, 2009), and Electronic patient record software (Martin, Rouncefield, O'Neill, Hartswood, & Randall, 2005). While many argue that functional requirements can be made generic and that the same software thus can successfully serve different organizations (Pollock & Williams, 2009), usability is well documented as a major challenge in such software implementations (Martin, Mariani, & Rouncefield, 2007; Wong, Veneziano, & Mahmud, 2016). Usability refer to how usable a system is for the users in terms of efficiency, effectiveness, and user-satisfaction (ISO, 2018). For a system to be usable, a common argument is that the user interfaces (UIs) should be designed based on the existing practices, understandings, and mental models of the intended user group (Martin et al., 2005; Rosson & Carroll, 2009). There is thus a strong relationship between usability, users and the context of use, and there are many reports from generic software implementation projects where end-users are left with complicated UIs that fits badly with established practices (e.g., Koppel et al., 2005; Topi, Lucas, & Babaian, 2005; Wong et al., 2016).

In our empirical case, based on an ongoing Action Research project, we have observed similar mismatch between design and work practices. Our focus is on a generic health information software called DHIS2. Over the last two decades, the software has moved from a domain and organization-specific routine reporting system for health indicators implemented in a few countries to a generic software platform, designed to be used in any case of health data reporting, analysis, and presentation. In this regard, the software is highly successful with implementations in over 80 countries in domains

such as disease surveillance, patient follow-ups, health commodity ordering, and logistics management. While the software has shown remarkable flexibility in supporting highly varying functional requirements, usability remains a persistent challenge in many of the implementations. This is especially prominent when the software is implemented to be used in radically different situations from what was initially intended, which is increasingly the case. For instance, in new sub-domains of health with different domain specific procedures, terminologies and conceptual logics (Nielsen & Sæbø, 2016). As experienced by end-users, problems typically take the shape of complicated UIs with abstract and unfamiliar terminology, structured in a way that provides little similarity to existing practices.

As what makes sense to users may vary significantly across domains, countries and organizations, design to ensure usability cannot only happen at the global level of development, during what we term *generic-level design*. It must be addressed also on the level of implementation, through the process of *implementation-level design*. Thus, a challenging and reoccurring question related to DHIS2 is how to improve usability in the implementations of this software? In order to answer this, we need a better understanding of the nature of usability design in generic software projects, what makes key challenges, and a vocabulary suited to describe them and how to deal with them.

Existing research around generic software usability has mainly been concerned with identifying usability-related problems, often subscribed to misfits between the software UIs and existing practice and users' mental models (Atashi, Khajouei, Azizi, & Dadashi, 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005; Topi et al., 2005; Wong et al., 2016). A few papers also illustrate how solutions to such issues are difficult due to conflicting priorities among project managers and the limited ability to shape the generic software as desired during implementation (Li, 2019; Martin, Mariani, & Rouncefield, 2004; Martin et al., 2007). In line with e.g., Dittrich (2014) and Dittrich et al., (2009), we argue that generic software implementation projects represents a significantly different enviroment for design and development than the typical in-house and product development projects in which methods for usability design are based. To advance research in this area, an explicit analysis of what characterizes design affecting usability in generic software is needed, which allow further research to address the aspects that could help to strengthen it.

In this paper, our aim is to address this gap by developing a conceptual framework to describe the usability-related design that unfolds on the generic and implementation level, and, based on this, discuss how usability can be strengthened. We base our framework on the concepts of *design for use*, and *design for design* (Ehn, 2008). Applying the framework on our case illustrates its relevance and enable us to say something about where effort could be put in with the overall aim of improved usability. Concretely, our research question for this paper is; *what characterizes design for usability in the implementation of generic software packages?* Through this understanding, we identify 'meta-usability' as an important aspect to the strengthening of usability. That is, how usable the generic software and other design-resources leveraged upon during implementation-level design are for ensuring usability. With this, we provide two contributions by 1) introducing a conceptual framework useful in analyzing and describing the nature of usability design in generic software projects, and 2) identify and discuss 'meta-usability' as an aspect of particular importance, which we argue should be subject to further research.

2 Related Research: Generic Usability and Customization

A system with good usability enables the intended users to achieve specific goals with effectiveness, efficiency, and satisfaction (ISO, 2018). This means that usability is not a result of the layout and structure of the UIs of a system alone, but how well these aspects work in relation to a particular set of users in a specific context of use. More concretely, usability is to us how well the UIs of software fits with existing practices, routines and mental models of the end-users (Norman, 2013; Rosson & Carroll, 2009). The tight relation to a specific set of users makes usability especially challenging in the context of generic software packages. While some aspects of design can follow universal principles

(Grudin, 1992; Norman, 2013), varying practices, terminology, existing technologies, and culture suggests that one-solution-fits-all cannot be achieved (Soh, Kien, & Tay-Yap, 2000).

There is thus a tension between systems being both *generic* and *usable,* as the first emphasizes the general, and the latter the specific. As articulated by Norman (1998, p. 78); *"making one device try to fit everyone in the world is a sure path toward an unsatisfactory product; it will inevitably provide unnecessary complexity for everyone"*. An array of literature has assessed and provided detailed accounts of usability problems in implemented generic software packages, often in ERP-systems (Topi et al., 2005; Wong et al., 2016), and in generic health software (Atashi et al., 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005). Reporting from the implementation of a generic ERP solution used in several countries across the globe, Topi et al. (2005, p. 132) provides a colorful and aptly quote from a frustrated end-user:

> *"it was like the spaceship had landed, and these outer space creatures [trainers] got off, and started talking to us about how we were going to do our job, because nobody understood what they were saying. Now, they're talking about notifications, material numbers, document control, material masters -- you know, that wasn't in any of our language"*.

A well-established means of making usable UIs is through the involvement of end-users in the process of design (Baxter & Sommerville, 2011; Kujala, 2003; Rosson & Carroll, 2009). Here, the nature of a generic software development project differs significantly from bespoke development practices. While development of context-specific software can emphasize local particularities of practice in design, it is near to impossible for developers of generic software to directly involve end-users and cater for the specifics of local practices across all implementations (Titlestad, Staring, & Braa, 2009). Thus, during implementation, generic software will typically need to be customized to the specifics of the use-case (Dittrich et al., 2009; Martin et al., 2004).

2.1 Customization during implementation

From a functional perspective, many see customization as an unwanted activity that complicates implementation and interfere with the ability to keep the software updated with new versions of the generic software. As such, customization is only to be a last resort if organizational practices are unable or "unwilling" to change. For instance, Light (2005) outline limited competence in the implementation team, and strategic motives such as maintaining the relevance of in-house developers as prominent rationales for local customization. Rothenberger and Srite (2009) subscribe the need for customization to factors such as users resistance to change, the implementer team's lack of authority in these manners, and their *"lack of opposition to customization requests"* (Rothenberger & Srite, 2009, p. 663). From such a perspective, usability receives limited focus, and users hesitant to change are to be blamed for problems associated with use.

Representing a more "user-friendly" strand of research, Dittrich (2014) acknowledge the need for customization during implementation if sufficient fit between software and organization are to be achieved. She argues that customization should rather be encouraged by designing 'half-way products', where local "customization development" is facilitated. Such design during implementation is however not straightforward. Martin et al. (2007) provide a detailed account of the process of implementation or "domestication" of a generic software. Often, usability problems are well known, but due to obstacles such as limited "tweakability" of the software and competition with other more functional requirements, they are not solvable. Martin et al. note that *"when straightforward technical solutions to usability problems cannot be found, they are inevitably turned into training issues"* (Martin et al., 2007, p. 55), resulting in situations similar to that of the end-user quoted above. Along the same line, Li (2019) argue that working with generic software during health software implementations represent an obstacle to ensuring usability as it may constrain the ability to design according to local needs. Martin et al. (2005) describe implementation work as an integration process, where software packages should be integrated with existing practices to be usable. The authors report from the implementation of a "customizable-of-the-shelf" system and illustrate how designers are

faced with the choice of addressing usability-related problems in the system through customization (technically), or through the change of user practices and training (socially). In their words, an essential factor is thus, *"how much it [the software] will have to be tweaked, and how tweakable it is"* (Martin et al., 2007, p. 48).

The issue of sufficient design flexibility to shape generic software according to the practices of local organizations is presented as a core issue by several researchers. Krabbel and Wetzel (1998, p. 46) present the nature of generic software implementation as a major challenge to user involvement activities, and Participatory Design more specifically. In the authors' view, the customization process on the level of implementation is of equal importance to that of traditional software development, but *"management [are] often misjudging this situation expecting an easy system implementation"*. The authors note that *"... systems can differ in their degree of flexibility to be adapted to the needs of the users. If adaptability is missing it means that the vendor has to change the system code for this customer"*. In other words, to achieve fit, and to respond to feedback from end-users, flexibility for adaptation or customization at the level of implementation is argued to be of essence. Similar arguments are made by Wulf, Pipek, and Won (2008), and Roland, Sanner, Sæbø, and Monteiro (2017, p. 8) arguing that *"end-user participation in development and adaptation of a software product can be enabled or constrained by the level of flexibility with the software itself"*.

We can see that usability problems of implemented generic software packages are frequently reported in the literature. While customization by some is presented as something to be avoided, as usability is tightly related to the specifics of each use-case, and global developers are unable to design UIs that fits all organizations, much research argues for the need of design on the level of implementation. This is not straightforward due to limited priorities of usability-related aspects at this level, and as designers deals with a pre-designed artifact, possibly constrained by the flexibility to shape it according to local practices.

3 Theoretical lens: Two levels and types of design

From the existing literature, we can conclude that design relevant to the usability of generic software is related to both the level of 'global' development, and the level of implementation where customization takes place. Based on this general understanding, we will in this section develop a conceptual lens for our analysis. To refer to the usability-related UI design processes on the two levels, we introduce the terms *generic-level design,* and *implementation-level design*. On both levels, different types of design unfold. To describe these, we adopt the concepts of '*design for use before use'* and *'design for design after design'* (Ehn, 2008), or more simply; 'design for use' and 'design for design'. The latter often referred to as meta-design (Fischer & Giaccardi, 2006). First, we will give a brief definition of our two levels of design, before relating them to the two types of design processes.

3.1 Generic-level and implementation-level design

We use the term *generic-level design* to refer to the design process unfolding during the development of the generic software product. This type of design and development has similarities to that of product development, where the emphasis is on creating a product to be used by a large audience (Grudin, 1991). Functionality and corresponding UIs are thus developed based on the anticipated need of this audience. However, as it is often recognized that both functional and non-functional requirements may vary, generic-level design also concerns the development of features and resources that allow customization of the product. In the words of Dittrich (2014, p. 1454) *"part of the design is deferred to other actors closer to the concrete use context"*. During *implementation-level design*, that is, the design process unfolding during the implementation of the generic software product, these features are leveraged upon to adapt the software to meet local user needs. At this level, design and development resemble that of in-house development (Grudin, 1991), however, with a basis in the traits of the generic software package at hand (Dittrich et al., 2009). Design is as such about integrating the software with local practice (Martin et al., 2005), and central to the designers or 'implementers' task is

to mediate between capabilities of the software, and the needs of the end-users and other actors of the implementing organization (Dittrich et al., 2009).

3.2 Design for use and design for design

The design unfolding on the two levels can be categorized into two types. *'Design for use'* refers to design-activities that unfold before a new or updated artifact has been introduced in a working stage to the intended end-users (Ehn, 2008). Many widespread design methodologies are based on this principle, such as User-Centered Design (Norman, 2013), and Participatory Design (Bratteteig, Bødker, Dittrich, Mogensen, & Simonsen, 2012). In the process, designers attempt to understand the users' current needs and practices, and predict and anticipate how the artifact to be designed can fit into this context. Both generic-level and implementation-level design entails this type of process. At the generic level, this regards the generic UIs that will be delivered to the end-users without any customizations during implementation. Moreover, at the level of implementation, the customization-based design could be categorized as design for use.

A common critique of this type of approach, which is highly relevant to generic-level designers, is the limitations in trying to anticipate use before it actually unfolds (Ehn, 2008; Fischer & Giaccardi, 2006). As both technologies, users, and context of use are ever-changing and evolving, this predictive form of design runs the risk of making systems that quickly become irrelevant, or without a sufficient fit from the start. Based on the limitations of design for use, '*design for design'* is based on the idea that software should be designed as open and flexible systems, or "half-way products" (Dittrich, 2014), allowing further design at a later stage. For generic software, this design after initial design takes place during implementation-level design. In practice, this means that the role of the generic-level designers is not to provide a finished product, but rather a *design infrastructure* (Ehn, 2008) providing implementation-level designers with the means of continuing to shape the artifact before final use. Design infrastructure refers to both technical and social resources that may enable design after the initial design. The infrastructure is thus not merely technical but could encompass all types of social and material elements that would aid design at a later stage (Fischer, 2008). Table 1 illustrate the relation between the levels and types of design.

	for design	for use
Generic-level design	X	X
Implementation-level design		X

Table 1. Relation between the levels and types of design in our framework

Design for design has most popularly been conceptualized in Fischer and Giaccardi (2006) framework of 'Meta-design'. The framework has mainly been applied in research on end-user development (EUD) (Ardito, Costabile, Desolda, & Matera, 2017; Fischer, Fogli, & Piccinno, 2017). The rationale behind EUD is generally to empower end-users with the tools needed to customize or extend the software themselves during use-time. From a usability point of view, this is an ideal situation as the designers are actual users able to shape the technology to correspond to their world. This form of development poses a somewhat different situation than in the implementation of generic software packages. As pointed out by Fogli and Piccinno (2013), it is often the case that end-users are not interested in engaging in customization and development work directly. Furthermore, these users typically have significantly varying computer skills. From a software governance point of view, having hundreds or even thousands of different customized versions of the software in circulation will also imply difficulties with user support, training, and system updates. Accordingly, the utilization of the design infrastructure in the case of generic software implementation happens during implementation-level design, rather than during end-use. For usability, this again means that designers are not users, and that issue of usability design is relevant (Fogli & Piccinno, 2013, p. 421). Thus, implementation-level design is about design for use based on the infrastructure provided by the generic-level designers.

To summarize our conceptual framework, generic-level design entails design of generic UIs for use, and design for design by building a design infrastructure to support customization on the level of implementation. Closer to the use-context, implementation-level design adds another process of design for use, before the product is used by the end-users. Table 2 summarizes these levels and relevant types of design. After presenting the methods for data collection, we will apply this conceptual framework on our empirical case, which illustrates its relevance and helps us identify where effort could be put in to ensure a more usable generic software for the end-users.

Level of design	Definition	General aim	Types of design
Generic-level design	The design process unfolding during the development of the generic software product	Support a variety of use through generic interfaces and customization features	Design (of generic UIs) for use and design for (implementation-level) design
Implementation-level design	The design process unfolding during the implementation of the generic software product	Appropriate the generic software to particularities	Design for use by leveraging upon the design-infrastructure built and maintained by generic-level design

Table 2. Definition, aim and types of design for the two levels of our framework

4 Methods

Our empirical case reports from an ongoing Action Research project concerned with health information systems development and implementation. Action Research is a methodology that allow researchers to understand organizational problems and attempt interventions to evaluate their effects (Baskerville & Wood-Harper, 2016). The process is cyclic, including phases of problem diagnosis, intervention planning, doing the intervention, evaluating the effects, and documenting the learnings. The project, called the Health Information Systems Programme (HISP), has over the last two decades been engaged in activities in a variety of developing countries (Braa, Monteiro, & Sahay, 2004). A central part of the project is the development of the generic software of focus in this paper, the health information software 'DHIS2'. When implemented, the software allows for the collection, storage, analysis and presentation of health-related data. To support implementation and continuous development of the software, an extensive network of nodes of local implementers has been established in countries such as South Africa, Tanzania, Uganda, and India. The nodes possess the required competence to configure the software to the data input and output needs in the use-case at hand. Within this network, there is ongoing research on several topics concerning systems development, integration, user participation, and ICT for development.

The authors of this paper have participated in the project several years, and been involved in activities at the global level of development as well as local implementations in many countries, including Uganda and India. Data for the concrete topic of this paper is collected through the diagnostic phase of a more specific Action Research initiative aimed at strengthening the usability of an implementation of DHIS2 in a state in India. The project was triggered by the implementing organizations explicit request of strengthened usability in their system. This allowed us as researchers to follow the HISP India team in diagnosing the usability problems experienced, and obstacles and possibilities for addressing these in the DHIS2 software. Related to these aspects, we have been in continual dialog with HISP India for several months, including spending a total of six weeks in the HISP India office, and on field-trips and meetings with end-users and managers in the implementing organization within the state. The experiences in India has simultaneously been discussed with generic-level designers. The project is now moving to the stage of action planning, where the findings from the diagnosis, partly presented in this paper will serve as a basis. Methods for data collection includes interviews, attending meetings, focus groups, and participatory observations at both the level of generic-level design and implementation-level design. Our aim is to improve our understanding of the overall

process of design related to usability, and more specifically how local implementers and developers work, and the challenges faced when appropriating the software to local conditions. Table 3 summarize data collection with different actors and through various activities.

Actors	**Activities**	**Number of participants**
Global developers	Informal interviews and attending meetings and discussions	Approximately 6
Local implementers and developers in India	Formal and informal interviews (approximately 6), focus group, participation in design, planning, and development activities (approximately 4 months), attending meetings (approximately 10 meetings).	8
Project managers in implementing organizations	Attending meetings and discussions (3)	8
End-users (data entry operators and health managers) in the implementing organization	Focus groups (4)	5

Table 3. Summary of data collection

During the engagement in the project, data has continuously been analyzed through a hermeneutic process of documentation and reflection (Klein & Myers, 1999). Concretely, principles and techniques from thematic analysis (Braun & Clarke, 2006) has been applied to code and categorize notes taken during data collection. Further, codes and categories has been grouped into themes, and their relation have been drawn in figures and thematic maps. Throughout the process, these themes have been discussed in light of, and linked to the existing literature presented in the previous chapters.

5 Case and Analysis

Our empirical case concerns the implementation of DHIS2 in a large (estimated population of 200 million) state in India. Referred to as the 'HMIS portal', the system is mainly serving routine health data reporting from districts to higher levels. End-users are mainly health managers and data entry operators on the various levels. In this section, we will use our theoretical lens of levels and types of design to understand what affects the achievement of usability during design of the UIs of the software. We will first provide a rather general account of the generic-level design of the software before we move into more detail on the process of implementation-level design.

5.1 Generic-level design

The generic-level design of the DHIS2 software is performed by a team referred to as the *core developers*. These include about thirty people, in the roles of designers and software developers, mainly situated in Oslo, Norway. Their aim is to build a software that supports variety so that it can be implemented to serve different types of requirements and use-cases. A central part of this is a configurable generic software 'core' where organizational structures, data elements, and relations are configured during implementation. Also, a set of *bundled apps* are developed as standard alternatives for users to perform data entry, analysis, and presentation. The design of the generic UIs of these apps can be seen as *design for use*, as many aspects will eventually face the end-users as designed on the generic level, thus directly affecting the usability.

5.2 Design for design

Many aspects will vary greatly between use-cases, so empowering the design process that ideally will take place during the implementation phase with flexibility for customization is seen as important. To this end, the core developers implement customization features in the software to enable implementers

to configure the software according to local needs. This includes features for configuring the generic core, and customizing the UIs of bundled apps. Furthermore, an application programming interface (API) is maintained to allow external parties and implementers in local projects to build custom *apps* using HTML, CSS, JavaScript and front-end frameworks. Finally, a 'dashboard' application is part of the generic DHIS2 package. The dashboard allows end-users to add and arrange the content of particular interest at the landing page of the software. Content can include links to applications and reports, graphs, maps, and tables. The design of the generic customization features, the API and the dashboard can be viewed as *design for design*, as the purpose is to allow further shaping of the application during implementation-level design.

To build capacity for the utilization of these technical features, extensive documentation is available online, and an educational certification system has been developed. This is called the 'DHIS2 Academies' and are arranged as regional conferences several times a year around the world with topics such as "Design and Customization", "Data use", and modules that are more specific to concrete aspects of the software. Members of the HISP network arrange academies locally and help to strengthen and share knowledge about best practices of implementation across projects. From an institutional perspective, the training resources and the DHIS2 academies also contribute to the design infrastructure built to support local customization. Furthermore, generic-level designers receive their requirements for further development and maintenance through the network of HISP-nodes. The requirements are mediated through digital channels such as email-lists and Jira, and through meetings with a consortium of "expert" local implementers on a regular basis.

To summarize, generic-level design of the DHIS2 software involves both design for use through the development of generic UIs used directly in a variety of use-cases, and design for design of the socio-technical infrastructure provided to the implementers.

5.3 Implementation-level design

In the implementation of focus in our state in India, the DHIS2 is configured to support state-wide reporting of routine health data. The local node *HISP India* is in charge of the implementation, working together with the implementing organization in the state. Several of the implementers in HISP India has attended DHIS2 academies, and are frequent users of the learning resources available online. They are as such utilizing the social component of the design infrastructure around DHIS2. The system implemented in India is referred to as the 'HMIS portal', and consist of a collection of generic DHIS2 components and a few custom-built apps developed by HISP India to support functional requirements that were not supported by the generic apps available. The generic components include a dashboard presenting particularly relevant graphs and tables of data per user-group, and the bundled apps for data entry, analysis, and presentation. Thousands of health facilities are using the system to report, analyze and present routine health data.

The process of implementation-level design for use has been ongoing through three "phases" since 2015. As the implementers are not actual end-users, the initial phase involved participatory activities aimed at establishing the data reporting requirements. That is, what data needs to be collected to satisfy the information need of the different actors involved, such as health managers at the district and state level. Based on this, data sets of around 4000 elements were defined, data entry forms were created and various output formats configured. During the initial phase, the system was introduced to the users throughout the state, and training sessions organized. As feedback and new requirements emerged, a new phase of customization and further development was taken on. A wealth of emerging requirement and limited time to meet them has been a characteristic of the implementation process. In the words of a HISP India implementer, *"requirement that comes on Friday, should be done on Monday"*. The project coordinator elaborates on this explaining that *"If the [customer] likes it, they want it right away. They come up with all sorts of reasons for why they need it quickly"*. This puts a lot of pressure on the implementing team to deliver new functionality and updates with limited time for thought on UI aspects. What data to be reported, and how to present this in the various bundled apps for data presentation was at the focal point of design in the two first phases. Technically,

requirements related to this has been relatively easy to meet through configuration of the core, and the bundled apps of DHIS2. Where appropriate functionality was lacking, custom apps have been built.

In a newly initiated phase three, the usability of the UIs of the implemented software has received explicit attention by the implementing organization. Feedback from frustrated end-users and managers triggered this phase, where typical problems identified include inconsistent interface design, complicated UIs, and frequent mismatches between the conceptual models of processes and terminology in the system UI and in the existing use practices. According to a project manager, this is particularly apparent in the amount of training they have to arrange with the end-users. Often users have to be *"re-trained several times on the same modules. It takes a lot of effort"*. Based on this, the implementing organization has provided a list of issues to be solved, and a set of suggestions for new layouts. The HISP India team has further explored these through focus groups and discussions with end-users at district health offices and other relevant locations. Moreover, an external usability expert has reviewed the UIs of the entire portal and provided comments on pressing issues. Summarizing these inputs, four areas have been given particular focus:

- Creating new dashboard content based on end-user needs. For instance, a way for data entry workers to easily see upcoming deadlines for reports.
- Improving the UI of the bundled app 'Pivot table', by removing unused menu options. The Pivot table is commonly used across implementations of DHIS2 by end-users to create different tables of data for analysis and presentation.
- Localizing the terminology of all apps in the portal to better align with terms familiar to the end-users.
- Making the design within the portal more consistent between apps (layout of buttons, lists, menus, etc.).

5.4 Design for use

Ideally, from a usability perspective, solving the issues outlined will require to localize the generic properties of the software to match the specific community of practice, and especially the UIs. As the HMIS portal is based on a generic software, doing so is not straightforward. Rather, the implementers have to find ways of leveraging on the material properties of the design infrastructure.

For DHIS2, the technical design infrastructure gives the implementers the choice of adapting the software UIs to local needs through three approaches: 1) 'generic customization', by configuring what is possible in the generic bundled apps. This was the main approach to design in the two first phases of the implementation project. 2) 'Forking' these apps, that is, downloading the source code openly available online and changing it as desired, and 3) developing custom apps using the API. Each of these approaches has both benefits and challenges. Generic customization is by far the fastest, most efficient, and least competence intensive choice. Albeit, flexibility is seen as limited in terms of UI design, which may constrain the ability to ensure sufficient usability. 'Forking' of apps gives more UI design flexibility but will require extensive software development competence and time. Further, future updates by the generic development will not be included in the forked app, and making the forked version of the app work with new versions of the software package may imply additional maintenance work. On this basis, forking apps are not seen as a good alternative. As argued by a HISP India implementer; *"the more we use generic functionality the less hassle with updating to new versions"*. Finally, developing custom apps gives the implementers extensive flexibility to design the UIs as preferred by the end-users. On the downside, the development of such apps is time-consuming and require competence as all functionality has to be built from scratch. Table 4 summarizes the pros and cons of each approach.

Technical choice	Benefits	Challenges

Configuration of bundled apps	Fast, easy	Limited design-flexibility
'Forking' bundled apps	High design-flexibility, provides a starting-point in terms of functionality as based on already working app	Time and competence intensive, need to understand existing codebase, maintenance work with software updates
Building custom apps	High design-flexibility	Time and competence intensive, has to build everything from scratch

Table 4. Technical features of the design infrastructure

As mediators between the design infrastructure on one hand, and the end user's needs on the other, the HISP India team have started the process of addressing these issues and suggestions for improvements by discussing how these could be catered for technically in the software. As the HMIS portal consists of a combination of the DHIS2 dashboard, generic bundled apps, and custom apps developed locally, the design flexibility is varying. For instance, the generic dashboard app is highly customizable. Pre-defined components such as graphs and maps can be added and arranged on screen without the need of programming. In addition, custom 'widgets' can be created using HTML, CSS, JavaScript and APIs. This allows the implementers to address the posed problems and suggestions with relative ease. For this particular challenge, addressing the needs of the end-users and ensuring usability seems feasible, and the HISP India team is now working to create widgets. The idea is that end-users later can choose from these widgets and arrange them on the screen as desired. In contrast, the changes required to make the 'Pivot table' app more usable would involve customization beyond generic configurations, which leaves the option of forking the app. With the issues related to future updates and required time this is not seen as a viable option. The problems related to consistent design and terminologies poses an even greater challenge, which spans both generic and custom apps. For custom apps built locally, UIs can relatively easily be modified with consistent design elements and a terminology suited for the end-users. For generic components such as the Pivot table, the data entry app, and other apps for data analysis and presentation, the implementers are at the mercy of the generic configuration features available, if they want to avoid the demanding process and challenges associated with forking or custom app development.

How to proceed with addressing the issues affecting usability in India is yet to be determined. With the dilemmas and difficulties faced, it is likely that many of them will be solved through more end-user training, rather than by attempting to design the UIs to better align with existing practices.

5.5 Summarizing the Characteristics

In sum, we have seen that generic-level design of DHIS2 entails both design for use and design for design. Implementation-level design, which in our case is performed by HISP India, concerns design for use. During implementation phase one and two, DHIS2 was at large able to support the rapidly emerging functional requirements by using the generic configuration options, and development of a few custom apps. However, the process of making the UIs usable based on existing practices are faced with multiple obstacles and dilemmas as each technical choice implies significant pros and cons. While the standard configuration features are quite limited, extensive design flexibility in 'forked' or custom apps development comes at the cost of time, resources, competence and constrained updatability. As such, the implementation-level design is highly affected by the design-infrastructure it operates within. Generic-level design is thus highly relevant to usability both directly and indirectly; 1) directly through the generic UIs used by end-users, and 2) indirectly, through the design infrastructure forming the basis for implementation-level design, which will enable or constrain the implementers in localizing the software sufficiently on their behalf. Figure 1 summarize the levels and types of design.

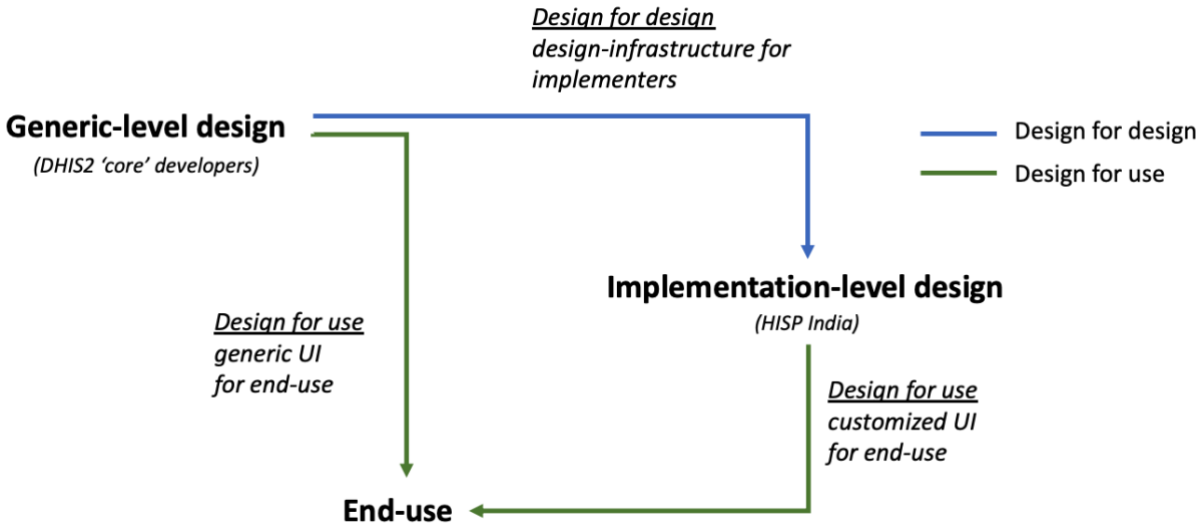

Figure 1. The two levels and types of design affecting usability

6 Discussion

In this section, we will discuss the characteristics identified, and based on this emphasize 'meta-usability' as particularly relevant to the strengthening of usability in generic software.

The end-users of the HMIS portal in India struggles with typical usability problems due to mismatches between UIs and existing work practices in terms of terminology, irrelevant menu options, and inconsistencies in UI layouts. This resembles challenges reported in prior literature on generic software implementations (e.g., Atashi et al., 2016; Khajouei & Jaspers, 2010; Koppel et al., 2005). Beyond outlining these problems, the conceptual framework we have developed in this paper has enabled us to analyze the key characteristics of usability-related design of the generic software implementation in our case. A strength of the framework is that it provides an explicit language to describe two types of design on two levels that we see as relevant to usability. In our case, the generic-level designers do design generic UIs to be used by end-users across use-cases. However, as implementations such as the one in India are filled with a variety of particularities, the generic-level designers cannot seek to sufficiently support everyone. In line with e.g., Martin et al. (2005), much is depending on the 'shaping' of the software during the implementation-level design process. It is thus important that customization is not neglected as an unwanted activity as portrayed by one strain of research on ERP system implementation (e.g., Rothenberger & Srite, 2009), where eventual problems are subscribed to end-users unwillingness to change. Rather, thought should be put into how implementation-level design best can be supported by the software and other components of the design-infrastructure to achieve sufficient 'shaping'. This illustrates the relevance of generic-level design for design. For usability to be attainable, in addition to developing usable generic UIs, generic-level design must focus on building technical features and relevant competencies for the implementation-level designers.

To suggest an answer to the title of this paper; achieving usability is about *both* global and local design. During 'global' generic-level design, design affects usability directly through generic UIs, and indirectly through the ability they give implementation-level designers to shape the software locally. The implementers leverage upon this design-infrastructure to ensure fit. Usability design is as such a joint effort between the two levels.

6.1 Meta-usability

As discussed by both Li (2019) and Martin et al. (2005), difficulties associated with 'shaping' the software as wanted represents a major obstacle for usability design during implementation. If

designing for usability within this design-infrastructure is difficult as in our case, it may significantly halt the process, and solving them through end-user training becomes a more viable option. How implementation-level design can best be supported through strategic design for design on the generic level is thus a prominent factor in the strengthening of usability. From an overall perspective, we see this as a key aspect to assuring a usable software for the end-users, which should receive more focus from researchers and practitioners. Representatives from the software industry have made similar arguments. For instance, Tao Dong, a User Experience Researcher at Google, recently articulated that *"the more usable developer tools are, the more energy developers can spend on delivering value to their users. Therefore, the UX [user experience] of developer products is just as important as for consumer products"* (Dong, 2017). Being largely dependent on the design-infrastructure provided by generic-level designers, this is particularly relevant for implementation-level design. Emphasizing its importance, we coin the term 'meta-usability' to refer to how usable the elements of the design infrastructure are in regards to achieving usability during implementation-level design. Two types of meta-usability are of particular prominence in our case:

1) How usable the software is in regards to customization and 'shaping' towards local practice, which could be referred to as *'design-usability'*
2) How usable the methods advocated through learning resources of the design-infrastructure are to aid the process of design, which could be referred to as *'method-usability'*.

First, what we refer to as *design-usability* concern how well the software supports implementation-level designers in the process of adapting it to local particularities. In addition to Martin et al. (2005) and Dittrich et al. (2009), this is in line with Singh and Wesson (2009), which describe *the ability to customize* as an important heuristic of generic software. In our case, design-usability consist of 1) customizability - what is customizable, 2) the degree of effort needed to utilize such features, and 3) to what extent utilization will collide with other desired aspects such as updates and maintenance (Light, 2001; Sestoft & Vaucouleur, 2008). This is particularly visible in our case where flexibility associated with the customization of *bundled apps* are too limited, and thus makes it impossible to shape the UIs according to local needs. At the same time, the creation of custom apps provides significant, or almost endless flexibility, however, at the cost of time, resources and need of competence. Both approaches has strong limitations when it comes to design-usability, with similarities to the *Turing Tar Pit* (Perlis, 1982) and its invert discussed by Fischer (2008, p. 368). Customization of bundled apps provides an environment "*where operations are easy, but little of interest is possible"* while during development of custom apps *"everything is possible, but nothing of interest is easy."* Also, if certain customization features significantly impact the work associated with updating to new versions of the global software package, they appear less usable from the implementers' point of view (Light, 2001; Sestoft & Vaucouleur, 2008).

Second, implementers are not end-users, so methods and techniques need to be used to understand how UIs best should be designed to integrate well with the use-context. Examples could be usability-inspections, user-centered design techniques or scenario-based design (Rosson & Carroll, 2009). Accordingly, in our case, such methods were applied to evaluate the UI to identify usability problems, and to gain knowledge on the real end-users' challenges related to UIs, and how to solve them. The methods used have often been conveyed to the implementers through the DHIS2 academies and learning resources that make up the social components of the design-infrastructure. Communication of methods well suited to achieve usability for the specific generic software, which are relatively easily adoptable by the implementation-level designers could be an important part of the design-infrastructure. As discussed by Baxter and Sommerville (2011), *method-usability*, that is, the usability of the method applied to aid the design process is thus relevant. For instance, the method needs to align well with the customization features of the software, the nature of the project, and the competencies and goals of the involved actors. Table 4 provides a summary of usability and meta-usability.

Concept	**Definition**

Usability	How usable the software is to a specific set of end-users
Meta-usability	How usable the elements of the design infrastructure are in regards to achieving usability during implementation-level design. Meta-usability includes: ***Design-usability:*** How usable the software is in regards to customization and 'shaping' towards local practice. Hence, how well the software supports implementation-level design. ***Method-usability:*** How usable the method applied to aid the process of usability design are, or, again, how well the method supports implementation-level design.

Table 5. Usability and meta-usability

6.2 How to strengthen usability?

Based on our analysis, it becomes apparent that an attempt to strengthen usability is not merely a matter of generic or implementation-level design, but rather will require a holistic perspective and intervention. Holistic in the way that interventions need to consider both generic and implementation-level design, and design for design and design for use. Particularly, design for design on the generic level will need to cater for design for use at the level of implementation. For instance, for DHIS2, a relevant intervention could be to extend the generic configuration options in the software to allow for the translation of terminology in the UI to correspond to an end-user familiar language. Also, strengthening design-usability by creating customization environments that are flexible, yet efficient and easy to use will be a great improvement. These technical interventions must be accompanied by the definitions and teaching of methods that correspond to these particular capabilities, thus increasing method-usability. This *software-method alignment* is in our opinion a particularly important factor, which has received limited focus in existing research. In sum, we argue that seeing the process as a means of strengthening meta-usability could be fruitful, where interventions to improve design-usability and method-usability should be an integrated process.

7 Conclusion and future research

We set out to explore the question; *what characterizes design for usability in the implementation of generic software packages?* In our case, the design process that affect the usability for end-users is characterized by two types of design (design for use and design for design) unfolding on two levels (generic-level design and implementation-level design). For usability to be attainable, generic-level design needs to focus on design for design, by building technical features and relevant competencies for the implementation-level designers. Implementation-level design needs to focus on design for use by mediating between the end-users existing practices and understandings, and the technical features of the generic software. Furthermore, strengthening usability will require a holistic intervention that involves both levels and types of design, where the generic-level designers' focus should lie on the strengthening of meta-usability. That is, ensuring that the elements of the design infrastructure are usable in regards to achieving usability during implementation-level design. This especially involves the alignment of software customization features and usability design-methods. Based on this, we suggest two topics suited for further research, which also are highly relevant to practitioners involved in generic software projects.

How to strengthen the design-usability of generic software?

This will require extended consideration of what needs to be customizable, how it can be made easy and efficient, while not interfering with the updatability from global releases. An interesting aspect is to find a balance between the "Turing tar pit" and its inverse. For this, a fruitful endeavor could be to explore the use of design-systems for app development to balance between flexibility, while keeping the barrier to take on such development as low as possible (Frost, 2016; Wulf et al., 2008).

How to strengthen the usability of design methods applied at implementation-level design?

Making methods more aptly would, as discussed, require and alignment between software and method, but also sensitivity to the nature of implementation projects and pressing issues such as scale, distribution and heterogeneity of users and practices within the "local" implementation level (Li, 2019; Sommerville et al., 2012).

References

Ardito, C., Costabile, M. F., Desolda, G., & Matera, M. (2017). A three-layer meta-design model for addressing domain-specific customizations. In *New Perspectives in End-User Development* (pp. 99-120): Springer.

Atashi, A., Khajouei, R., Azizi, A., & Dadashi, A. (2016). User Interface problems of a nationwide inpatient information system: a heuristic evaluation. *Applied clinical informatics, 7*(1), 89.

Baskerville, R. L., & Wood-Harper, A. T. (2016). A critical perspective on action research as a method for information systems research. In *Enacting Research Methods in Information Systems: Volume 2* (pp. 169-190): Springer.

Baxter, G., & Sommerville, I. (2011). Socio-technical systems: From design methods to systems engineering. *Interacting with computers, 23*(1), 4-17.

Braa, J., Monteiro, E., & Sahay, S. (2004). Networks of action: sustainable health information systems across developing countries. *Mis Quarterly*, 337-362.

Bratteteig, T., Bødker, K., Dittrich, Y., Mogensen, P. H., & Simonsen, J. (2012). Organising principles and general guidelines for Participatory Design Projects. *Routledge Handbook of Participatory Design*, 117.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.

Dittrich, Y. (2014). Software engineering beyond the project–Sustaining software ecosystems. *Information and Software Technology, 56*(11), 1436-1456.

Dittrich, Y., Vaucouleur, S., & Giff, S. (2009). ERP customization as software engineering: knowledge sharing and cooperation. *IEEE software*(6), 41-47.

Dong, T. (2017). Developer UX at Google. *Medium.com.* Retrieved from https://medium.com/google-design/how-i-do-developer-ux-at-google-b21646c2c4df

Ehn, P. (2008). *Participation in design things.* Paper presented at the Proceedings of the tenth anniversary conference on participatory design 2008.

Fischer, G. (2008). Rethinking software design in participation cultures. *Automated Software Engineering, 15*(3), 365-377.

Fischer, G., Fogli, D., & Piccinno, A. (2017). Revisiting and broadening the meta-design framework for end-user development. In *New perspectives in end-user development* (pp. 61-97): Springer.

Fischer, G., & Giaccardi, E. (2006). Meta-design: A framework for the future of end-user development. In *End user development* (pp. 427-457): Springer.

Fogli, D., & Piccinno, A. (2013). Enabling domain experts to develop usable software artifacts. In *Organizational change and information systems* (pp. 419-428): Springer.

Frost, B. (2016). *Atomic design*: Brad Frost.

Grudin, J. (1991). Interactive systems: Bridging the gaps between developers and users. *Computer*(4), 59-69.

Grudin, J. (1992). Utility and usability: research issues and development contexts. *Interacting with computers, 4*(2), 209-217.

ISO. (2018). ISO 9241-11:2018. In *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*: International Organization for Standardization, Geneva, Switzerland.

Khajouei, R., & Jaspers, M. (2010). The impact of CPOE medication systems' design aspects on usability, workflow and medication orders. *Methods of information in medicine, 49*(01), 03-19.

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *Mis Quarterly*, 67-93.

Koppel, R., Metlay, J. P., Cohen, A., Abaluck, B., Localio, A. R., Kimmel, S. E., & Strom, B. L. (2005). Role of computerized physician order entry systems in facilitating medication errors. *Jama, 293*(10), 1197-1203.

Krabbel, A., & Wetzel, I. (1998). *The customization process for organizational package information systems: A challenge for participatory design.* Paper presented at the Proceedings of the PDC.

Kujala, S. (2003). User involvement: a review of the benefits and challenges. *Behaviour & information technology, 22*(1), 1-16.

Li, M. (2019). *Usability Problems and Obstacles to Addressing them in Health Information Software Implementations*. Paper presented at the The International Conference on Social Implications of Computers in Developing Countries (IFIP WG 9.4), Dar es Salaam, Tanzania.

Light, B. (2001). The maintenance implications of the customization of ERP software. *Journal of software maintenance and evolution: research and practice, 13*(6), 415-429.

Light, B. (2005). Going beyond 'misfit'as a reason for ERP package customisation. *Computers in industry, 56*(6), 606-619.

Martin, D., Mariani, J., & Rouncefield, M. (2004). Implementing an HIS project: everyday features and practicalities of NHS project work. *Health Informatics Journal, 10*(4), 303-313.

Martin, D., Mariani, J., & Rouncefield, M. (2007). Managing integration work in an NHS electronic patient record (EPR) project. *Health Informatics Journal, 13*(1), 47-56.

Martin, D., Rouncefield, M., O'Neill, J., Hartswood, M., & Randall, D. (2005). *Timing in the art of integration:'that's how the bastille got stormed'.* Paper presented at the Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work.

Nielsen, P., & Sæbø, J. I. (2016). Three strategies for functional architecting: cases from the health systems of developing countries. *Information Technology for Development, 22*(1), 134-151.

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*: Constellation.

Norman, D. A. (1998). *The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution*: MIT press.

Perlis, A. J. (1982). Special feature: Epigrams on programming. *ACM Sigplan Notices, 17*(9), 7-13.

Pollock, N., & Williams, R. (2009). Global software and its provenance: generification work in the production of organisational software packages. In *Configuring User-Designer Relations* (pp. 193-218): Springer.

Roland, L. K., Sanner, T. A., Sæbø, J. I., & Monteiro, E. (2017). P for Platform: Architectures of large-scale participatory design. *Scandinavian Journal of information systems, 29*(2).

Rosson, M. B., & Carroll, J. M. (2009). Scenario-based design. In *Human-computer interaction* (pp. 161-180): CRC Press.

Rothenberger, M. A., & Srite, M. (2009). An investigation of customization in ERP system implementations. *IEEE Transactions on Engineering Management, 56*(4), 663-676.

Sestoft, P., & Vaucouleur, S. (2008). Technologies for evolvable software products: The conflict between customizations and evolution. In *Advances in Software Engineering* (pp. 216-253): Springer.

Singh, A., & Wesson, J. (2009). *Evaluation criteria for assessing the usability of ERP systems.* Paper presented at the Proceedings of the 2009 annual research conference of the South African Institute of Computer Scientists and Information Technologists.

Soh, C., Kien, S. S., & Tay-Yap, J. (2000). Cultural fits and misfits: is ERP a universal solution? *Communications of the ACM, 43*(4), 47-47.

Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M., . . . Paige, R. (2012). Large-scale complex IT systems. *Communications of the ACM, 55*(7), 71-77.

Titlestad, O. H., Staring, K., & Braa, J. (2009). Distributed development to enable user participation: Multilevel design in the HISP network. *Scandinavian Journal of information systems, 21*(1), 3.

Topi, H., Lucas, W. T., & Babaian, T. (2005). *Identifying Usability Issues with an ERP Implementation.* Paper presented at the ICEIS.

Wong, W.-P., Veneziano, V., & Mahmud, I. (2016). Usability of Enterprise Resource Planning software systems: an evaluative analysis of the use of SAP in the textile industry in Bangladesh. *Information Development, 32*(4), 1027-1041.

Wulf, V., Pipek, V., & Won, M. (2008). Component-based tailorability: Enabling highly flexible software applications. *International Journal of human-Computer studies, 66*(1), 1-22.

II

GENERIC ENTERPRISE SOFTWARE IMPLEMENTATION AS CONTEXT FOR USER-ORIENTED DESIGN: THREE CONDITIONS AND THEIR IMPLICATIONS FOR VENDORS.

Research paper

Magnus Li
Department of Informatics, University of Oslo
magl@ifi.uio.no

Abstract

User-oriented approaches to designing IT are consistently promoted by academic and practitioner literature. These orients the design process around the specific practices and needs of end-users to build usable and relevant systems. However, an increasingly relevant but little explored context for the design of IT is that of implementing generic enterprise software solutions. In this paper, we explore conditions for user-oriented design during the implementation of generic enterprise software. Our empirical data is based on an ongoing engaged research project, where we work with the vendor of a global generic software solution and a set of implementation specialist groups (ISGs). Together, we explore how user-oriented design during implementation of the software solution can be supported and promoted. The paper contributes to the body of knowledge on the design and implementation of generic enterprise software by identifying several challenges and three conditions for user-oriented design in this context. The conditions are: the project configuration, the implementation practices of the ISGs, and the features and adaption capabilities of the generic software solution. We further contribute by discussing their implications for vendors who want to support and promote user-oriented design during implementation of their software solutions.

Keywords: generic enterprise software implementation, user-oriented design, conditions, implementation-level design.

1 Introduction

User-oriented approaches to design and innovation, such as User-Centered Design and Participatory Design emphasize basing the design of IT on the practices and needs of specific end-users. These approaches are consistently promoted by research (Baxter & Sommerville, 2011; Ellingsen & Hertzum, 2019; Gulliksen et al., 2003; Mumford, 2006) and practitioner guidelines and literature (digitalprinciples.org, 2019; gov.uk, 2019; D. Norman, 2013). Meanwhile, a significant portion of the IT systems implemented in organizations are not built 'bottom up' based on the specific practices and needs of singular organizations. They are rather designed and developed as comprehensive generic software solutions that aim to serve a diverse audience (Berente et al., 2019; Pollock et al., 2007; Sykes & Venkatesh, 2017). Examples are Enterprise Resource Planning Systems (ERPs), and Electronic Health Record Systems (EHRs). Two increasingly relevant contexts for designing IT are thus that of building generic solutions (Pollock et al., 2007), and that of implementing these solutions into specific organizations by configuring them to local needs (Bansler, 2021; Dittrich, 2014;

Ellingsen & Hertzum, 2019; Martin et al., 2007). In this paper, we refer to the latter context as implementation-level design.

Traditionally, generic enterprise software solutions have been described as inflexible for local adaption, and the process of implementation-level design as one of adapting the organization according to the software, rather than building software according to their specific needs (Kallinikos, 2004). As a result, Information Systems (IS) literature argues, generic solutions often fail to meet the expectation of organizations and that the consequences for end-users and the organization as a whole may be adverse (Berente et al., 2019; Soh & Sia, 2008; Strong & Volkoff, 2010). However, in recent years, vendors of generic enterprise software are "opening up" their solutions for design and innovation by third-party actors (Farhoomand, 2007; Wareham et al., 2014). Design and innovation are no longer reserved for the vendor firm but supported and encouraged for a larger 'ecosystem' (Dittrich, 2014; Wareham et al., 2014) or 'design network' (Koch, 2007) of partner organizations that specialize in implementing the solutions on behalf of user organizations (Foerderer et al., 2019). Vendors move from building monolithic solutions or 'packages', to building platforms that are advertised as highly configurable and extendible to serve heterogeneous needs (Foerderer et al., 2019; Rickmann et al., 2014). This increasing emphasis on supporting design and innovation outside the boundaries of the vendor appears to offer the potential for more user-oriented design and innovation based on the needs of individual user organizations than what is earlier described in the literature. Still, to the authors' knowledge, no systematic analysis of the conditions for user-oriented design processes in the context of generic software implementation exists. Further, there is no literature examining how vendors may support and promote such design during the implementation of their solutions.

This paper addresses this gap by examining the following research questions:

- What conditions affect the potential for user-oriented design in the context of generic enterprise software implementation?
- What implications do these conditions have for vendors who want to promote user-orientation during implementation of their solutions?

We explore our two questions based on data collected through an ongoing engaged research project (Li, 2019), where we collaborate with a generic health software vendor and a set of implementation specialist groups (ISGs). The ISGs are independent consultancy firms that specialize in implementing the software for user organizations. The software solution, named DHIS2, is designed to support collection, and use of routine health information within organizations such as health ministries and non-governmental organizations. During the last two decades, the software has been implemented to serve a range of health-related use-cases and is now used by organizations in more than 80 countries. These user organizations have different practices and needs that, in many cases, would be best supported by IT solutions with custom functionality and user interfaces. Due to differences in needs, it is challenging for the vendor to design generic functionality and user interfaces that are considered usable and relevant across the vast audience of user organizations. A strategy the vendor increasingly pursues is that of supporting design and innovation based on specific organizational needs during implementation-level design. Part of this strategy is to make the solution configurable and extendible, and promoting the use of user-oriented approaches to design by the ISGs specializing in implementing it. In our work, we have, however, found that there are several conditions that make the implementation of a generic solution a challenging context for user-oriented design.

The rest of the paper is organized in the following manner: We first look at existing literature related to user-oriented design in the context of generic enterprise software implementation. We then describe our research approach before we present our analysis, where we examine the process of implementation-level design and challenges related to user-oriented design. In the discussion chapter, we articulate and discuss three conditions and their implications on vendors who seek to support and promote user-oriented design during implementation of their solution.

2 Related Research

The literature on user-oriented design in the context of generic enterprise software design and implementation is scarce, but with a few exceptions. We will first define what we mean by "user-oriented design" before turning to generic enterprise software implementation as a context for design.

2.1 User-oriented design

We employ the term 'user-oriented design' to refer to approaches to designing systems and technologies that orient the design process around the end-users needs and well-being, with the aim of making systems that are perceived as usable and relevant. A myriad of such approaches is conceptualized in IS and HCI literature. Readily available examples include User- or Human-Centered Design (Gulliksen et al., 2003; D. Norman, 2013), Participatory Design (Simonsen & Robertson, 2012), Activity-Centered Design (Gay & Hembrooke, 2004), Socio-technical Design (Mumford, 2006), and Usability Engineering (Nielsen, 1994; Rosson & Carroll, 2002). Although all are oriented towards the end-users of technology, they vary with regards to the ends and means of doing so, and the scope of what is to be designed (Kujala, 2003). For instance, Participatory and Socio-technical design are based on the idea that end-users should be involved in the decisions regarding the technology to be used in their work. Hence, a key aim of the process is to empower workers by giving them a voice in the design process. Means to achieve this naturally rely heavily on involving users in decisions regarding the IT project (Mumford, 2006; Simonsen & Robertson, 2012). In contrast to Participatory Design, the primary end of User-Centered Design is to build technology that is usable and relevant for end-users. Means of doing so do not necessarily include involving end-users in all decisions regarding the project, but often instead focus on understanding their existing practices and needs through interviews, observation, and iterative, evolutionary prototyping and evaluation (Norman, 2013; Norman, 2005).

Albeit differences in the ends, means, and scope of various user-oriented approaches to design, they share some key principles:

1. The features of technology are designed based on an understanding of the practices and needs of end-users in concrete contexts. Objectives of the design process are to establish 'what is the right thing to build' i.e., fundamental questions about the IT artifacts form and function, and 'building the thing right', i.e., defining the right form of the artifact (e.g., user interfaces, functionalities)
2. Iterative design and development with evolutionary prototyping and frequent end-user evaluations of form and function. Prototyping should ideally start with low-fidelity prototypes to ensure that the project avoids committing to a specific solution at an early stage. Rather, problems and multiple potential solutions are explored as the project evolves.
3. Emphasis on understanding the practices of and/or involving end-users in the design process, either in an informative role (as in User-Centered Design), or as active participants in fundamental decisions about the project and the artifact(s) of focus (Damodaran, 1996).

These principles form the basis for our understanding of user-oriented design. Accordingly, what we seek to identify in this paper are conditions that affect if, or to what degree, processes following these principles can take place. While many different user-oriented approaches to design are conceptualized, existing literature focuses little on the conditions that must be in place for such design processes to unfold (Baxter & Sommerville, 2011; Edwards et al., 2010; Svanæs & Gulliksen, 2008; Zahlsen et al., 2020). For instance, as noted by Svanæs & Gulliksen, (2008), the two ISO standards describing how user-centered design should be carried out *"describe an ideal situation where there are no obstacles to UCD, except for a possible lack of skills at the developer side. Although the two ISO standards on UCD are very useful as reference frameworks and ideals, they do not deal with the heterogeneous nature of real-world UCD projects, and the potential obstacles to user-centered design."* A few studies report how the 'boundary conditions' (Zahlsen et al., 2020) of the context where the design process takes place strongly impact the form of 'user-orientedness' that is relevant and possible. Such boundary conditions include internal factors in the developer organization, such as their structure,

software engineering practices, and 'usability maturity' (Earthy, 1998). Further, it may include aspects of how the project is structured with its actors, defined goals, and expected process (Martin et al., 2007). Others argue that a significant challenge is that user-oriented approaches are incompatible with widely used software engineering methodologies (Baxter & Sommerville, 2011).

2.2 Enterprise software implementation as the context of design

The development and implementation of generic enterprise software as a context of design differs from that of bespoke development (Li & Nielsen, 2019), often assumed by user-oriented approaches (Edwards et al., 2010). On the generic level of design, the vendor deals with significantly diverse and potentially incompatible needs when attempting to support a large audience of organizations (Sia & Soh, 2007). Design is reported to unfold as a process of aligning the needs of organizations seen as strategically important (Gizaw et al., 2017; Pollock et al., 2007), while neglecting needs that are relevant to only one or a few (Koch, 2007; Sia & Soh, 2007). Implementing the software into a particular user organization can be seen as another level of design, which we here refer to as implementation-level design (Li & Nielsen, 2019). During this process, the solution is configured and possibly extended according to the particular circumstances of the user organization (Sommerville, 2008). However, implementation-level design is based on the features and adaption capabilities of the generic solution, which do not provide endless flexibility for local adaption (Martin et al., 2007). Instead, the solutions are often adaptable in specific ways, dependent on the configuration facilities embedded by the vendor (Bertram et al., 2012). When generic features and adaption capabilities fall short, the source code of the software may be modified, but with the costs of additional work related to upgrading the software to new versions provided by the vendor (Hustad et al., 2016; Sestoft & Vaucouleur, 2008).

Research on user-oriented design in the context of generic enterprise software design and implementation is scarce, but with some exceptions. A few studies explicitly discuss user-oriented approaches such as User-Centered Design (Vilpola, 2008) and Participatory Design (Magnusson et al., 2010; Pries-Heje & Dittrich, 2009) as means of driving implementation-level design processes. However, these studies are more concerned with the use of such approaches to increase the user acceptance of the generic solution, rather than using them as the engine to design and innovate IT solutions based on insights into the end-users' particular practices and needs. There are few reflections on the conditions that affect the potential for conducting user-oriented design as we defined it in the previous section. Other studies discuss how design flexibility, as touched upon above, is a key challenge when addressing issues of usability and end-user relevance in implementations (Martin et al., 2007). Also, extendible platform architectures have been discussed as enabling "local" user-oriented design during implementation, as it allows custom applications to be built on top of the generic solution to address implementation-specific needs (Roland et al., 2017). Supporting custom app development also appears as a strategy followed by prominent vendors such as SAP to facilitate design and innovation during implementation of their widely used ERP solutions (Farhoomand, 2007; *SAP Fiori*, 2020)

To summarize, existing research emphasizes the importance of user-oriented approaches to design. Yet, one of the most common means of introducing technology in organizations is by implementing generic software solutions. We see the relatively limited knowledge on user-oriented design in the context of implementation-level design as an important gap in existing research.

3 Research Approach

We report from an ongoing engaged (Mathiassen & Nielsen, 2008; Van de Ven, 2007) research project (Li, 2019) where we collaborate with a software vendor – referred to as the 'core team' and a set of implementation specialist groups (ISGs). First, we briefly introduce some key information about the software solution and actors of focus before describing our methods for data collection and analysis.

3.1 Case – DHIS2, the core team, and the ISGs

We follow the generic health information software DHIS2. The "core team", situated in Oslo, Norway, is in charge of designing, developing, and maintaining DHIS2 as a generic solution. The DHIS2 is used by a diverse audience of organizations across more than 80 countries. In its primary use case, DHIS2 supports the collection, storage, and presentation of health management information. Due to its flexible and configurable data model, it is increasingly implemented for use in domains beyond health management, such as logistics management and education management. The generic solution comprises a software "core" with a configurable data model, and a set of "generic apps". The apps provide functionality and user interfaces that support activities common among end-users across implementations. Examples are reporting data and displaying information in reports, graphs, and maps. The core team decides what features to include in the generic solution by identifying shared needs across the user audience. Their means of doing so is beyond the scope of this paper. To support the adaption of the software when implemented in specific organizations, the core team embeds an array of configuration facilities into the software. This allows specific implementations to define certain aspects of the solution according to their particular circumstances. Examples include what data to report, when, where, and by whom, and how this data is to be presented in graphs, maps, etc. The generic solution is also extendible, meaning that so-called "custom apps" can be developed during implementation to extend the functionality and user interfaces beyond what is provided by the generic apps.

The ISGs we collaborate with in our study are independent consultancy firms that are contracted by (future) user organizations. Together they configure and extend DHIS2 according to their particular needs. The inner team of the ISGs typically includes what is called "implementers" in charge of working with the organization to identify their requirements and configure DHIS2 accordingly. The ISGs often also have a group of developers that build custom apps when needed.

3.2 Data collection and analysis

Our collaboration with the core team and the ISGs involves both diagnostic and interventionist research, and is interpretive in nature (Klein & Myers, 1999). As DHIS2 experiences increasing adoption by user organizations with diverse practices and needs, the usability and relevance of the generic solution is becoming an increasing concern. With this problem as the basis, we started exploring the nature and challenges of designing (with) DHIS2 during implementation. Our engagement started by participating as 'attached insiders' (Myers, 2019; Van de Ven, 2007) in a large implementation project in India together with an Indian ISG from August 2018 – November 2019, where we tried to address various usability issues that had been documented in the implemented DHIS2 solution. We participated in meetings (6), and in planning-activities, and conducted interviews (8) with stakeholders in the user organization, and the ISG team. Engagement in this project gave us insights into the process of implementation-level design and highlighted several challenges forming the basis for the findings presented in this paper. For instance, challenges found relate to how the defined scope of the project, and the mandate of the ISG therein restricted the relevance of and ability to interact with end-users to diagnose usability issues.

To get a richer understanding of the broader DHIS2 implementation practices beyond the Indian ISG, we continued to explore the nature of implementation-level design of DHIS2 by visiting three ISGs in Tanzania, Mozambique, and Malawi from May 2019 to January 2020. During these visits, which typically lasted for one week, we conducted interviews with implementation experts (6) and software developers (4). The interview subjects had 3 – 12 years of experience with DHIS2 implementation. We also arranged focus groups (3), including the whole or most of the ISG teams (6 – 12 participants) at their offices. The aim during interviews and focus groups was to understand the implementation-level design process, what activities it constitutes, and if, how, and when the design process is (not) oriented towards end-users. From March – November 2020, we also conducted interviews (4) with implementers and developers over Zoom with ISGs in the United States and Uganda, also focusing on

the process of implementation-level design and user-orientation in the activities it constitutes. Engagement in India, combined with interviews and focus groups with other ISGs allowed us to identify the common traits and differences between implementation practices and projects, and a variety of challenges they face related to user-oriented design. The findings are used as a basis to inform further diagnostic activities, and potential interventions in future phases of the research project. Data is collected and analyzed concurrently in our project. Data is documented through field notes (during participant observation and focus groups), transcriptions of interviews, and documents collected in the document analysis. An overall research diary is kept throughout the process, summarizing patterns and findings relevant to the various (and developing) research questions. The analysis is abductive in nature (Tavory & Timmermans, 2019; Van de Ven, 2007), comprising cycles of inductive analysis of empirical data (e.g., coding and developing themes related to practices and challenges which are presented in the case analysis chapter), and identifying similar phenomenon and related concepts in relevant IS literature (e.g., design and implementation of generic enterprise software, user-oriented design, enterprise software ecosystems).

4 Case Analysis

We now turn to our analysis of implementation-level design as a context for user-oriented design. We begin by looking at how the process unfolds, before we highlight some key challenges.

4.1 The process of implementation-level design

A typical implementation-level design process starts when the ISG is awarded a contract for a project following a tender process. The initial negotiation of how the project will be configured begins between the user organization, the ISGs, and potentially other involved actors. What we here name the 'project configuration' refers to how the project is defined in terms of the scope of the problem to be addressed and potential solution(s), structure and process of the project, and the mandate of different actors therein. The starting point in many projects is that an organization already has an existing digital or paper-based (or partly both) information system that supports the collection and presentation of some sort of data, often related to health management. The aim of projects of this kind is to design a coherent digital system based on DHIS2 to replace paper-based data reporting tools and by integrating various fragmented systems. Although the process of implementation-level design varies between ISGs and projects, we highlight five activities that typically make important parts of the process. These are illustrated in Figure 1, and we will use these to structure the first part of our analysis. We stress that these are not discrete steps of a linear process but rather activities that can be enacted several times, in various order, and often concurrently.

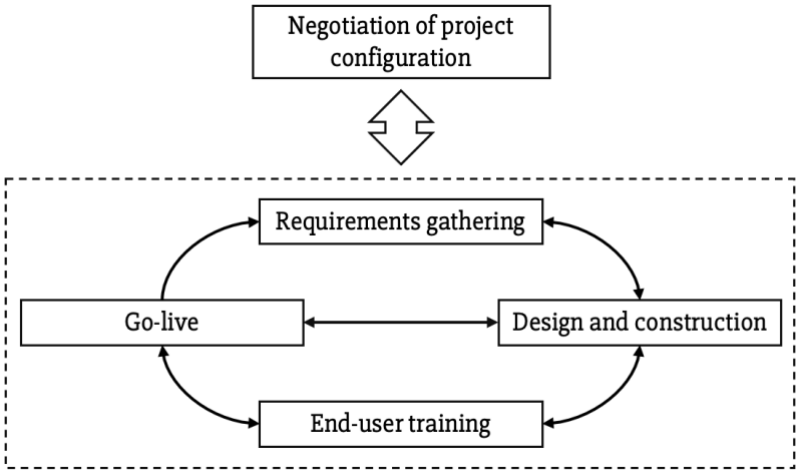

Figure 1. Five key activities part of the implementation-level design process of the ISGs

4.1.1 Initial negotiation of project configuration

Already in the initial negotiation of the project configuration, the generic features of DHIS2 play an important role in guiding and framing the discussions of the shape and form of the software solution for the particular user organization. The ISG tends to rapidly set up a running demo of DHIS2, including some configurations that show how the system could look and behave in the respective user organization. The ISGs explain that this is beneficial as it allows them to establish realistic expectations and show quick results early on, through what, at least, appears as a fully working prototype.

If, how, and which end-users are to be involved in the process is also partly established in the initial project configuration. Some ISGs explain how they push for making visits to health facilities and involvement of users in evaluations of the system part of the agreed implementation process. The relevance of engaging with end-users is often something that has to be advocated to the project managers of the user organization. For instance, the Mozambique ISG lead explains how he sees him and his team as *"fighting the battle on behalf of the end-users",* when negotiating if, when, and how end-users should be consulted and involved in the implementation process. In Malawi, they often try to negotiate the User-Centered Design methodology (as defined by the ISO standard) as a formal part of the implementation process. In contrast, other ISGs are not particularly concerned with promoting user-oriented activities, and prefer to primarily rely on communication with the project managers of the user organization. An implementer within one of the ISGs that falls under the latter category explains that user involvement is "painful" as the end-users seldom "agree" with the prototypes they show them. More so, he argues, users *"often quit their job even before the solution is launched anyway"*. The implementer explains how he sees end-user interaction as more about convincing the users to use the system, rather than for the end-users to give feedback for improvements – *"it's the [top level project managers] who decide anyway, it's them we have to please"*. He further explains that in the case of doing end-user interaction such as going to health facilities to talk to users, it's mainly when the project managers of the user organization ask for it explicitly.

4.1.2 Requirements gathering

The nature of how requirements are gathered and the role of the ISG in this activity differs substantially between projects. In some projects, the ISG only acts as a "technical partner" and merely a receiver of requirements defined by the user organization themselves or other consultancy firms. In other projects, the ISGs may be responsible for collecting, and defining the requirements throughout the development process. In the latter case, there is some variation among the ISGs and the projects on how requirements are collected and established. Some ISGs express the importance of doing extensive visits to the end-users' context, to *"map out current practices, tools, infrastructural conditions"* (Mozambique implementer), and other relevant aspects. As articulated by one implementer, *"If you're not doing it [field-visits], you go in blind [to the development process] […] you need to understand the context"*. Others rely exclusively on communication with project managers of the user organization.

Albeit differences in the means of requirements gathering, it is striking how similar the requirements gathering activity is in terms of what the ISGs look for. Either when visiting end-users in their context, and/or communicating with project managers, the focus is almost exclusively directed towards:

- what data is currently and/or in the future is to be reported, how and by whom
- what data is currently and/or in the future is to be used, how and by whom

"We identify the data elements, then we try to implement the data outputs and present them to the client […] We iterate between input and output several times" (Implementer India ISG).

The findings are documented along the requirements gathering process, and will normally be followed by new negotiations of the project configuration together with the project managers of the user organization (e.g., updating objectives and timelines).

4.1.3 Design and construction

The design and construction activity involves prototyping and constructing a working IT solution based on the collected requirements. In essence, this is about identifying how requirements can be accommodated by the generic features of the DHIS2, and if any custom development is required. Since most needs within projects are addressed by configuring DHIS2, the software is often used as a working prototype throughout the design and construction process. Regular evaluations of the prototype with representatives from the user organization are commonly carried out during the process to evaluate and adjust how requirements are to be addressed in the solution. Few of the ISGs involve end-users in this iterative process, and rather rely on frequent communication with the project managers of the user organization. A large portion of the process internally within the ISGs' technical team is to discuss how to configure the data model to best support the required data input and output. The configuration facilities of DHIS2 are described by the ISGs as highly supportive in this process, allowing for easy definition of organizational hierarchies, data elements to be collected, the layout of the reporting forms, and the various forms of data presentations that are needed.

When requirements and needs warrant changes or additions to functionality and user interfaces, the source code of the generic apps could be customized, resulting in a custom app for the specific implementation. Also, custom apps may be built from scratch. Through customization of generic apps, or by developing apps from scratch, the ISGs thus have extensive flexibility to shape and extend the generic features of DHIS2 with novel functionality and user interfaces. However, what typically limits the use of this flexibility is that it is costly to develop the apps (i.e., writing the code, designing the user interfaces, etc.) and maintaining them over time. An implementer reflects on this with an example from a recent project, where several modifications to the functionality and user interfaces were wanted, but avoided: *«If we had started to do modifications [to the generic features], we had lost the ability to update. That is, the benefit of being on the platform, and then you're suddenly alone. It gets difficult to update, and you cannot be part of getting new features together with the others.»*

4.1.4 End-user training and Go-live

ISGs tend to play a key role in end-user training. Users are gathered in workshops where they are trained to use the constructed solution. Some ISGs describe the activity as one of training and "convincing" the end-users in using the solution. Others regard the experiences from the trainings as valuable learnings related to the usability of the solution. Issues discovered may be addressed through more design and construction work. In this case, discoveries that could warrant more design and construction would often require a renegotiation of the project configuration. Finally, the solution is introduced for use in the organization, either starting with a small pilot with a few use-sites, or by introducing the solution to the whole organization all at once. The role of the ISG seldom ends here, and instead continue with maintaining and improving aspects of the solution, or providing "refreshment training" of end-users for several years to come. The further development of the system forms new cycles of the five activities outlined.

4.2 Some prominent challenges related to user-orientation

We now highlight some prominent challenges that came up during data collection and analysis.

4.2.1 Balancing flexibility and predictability

Fundamental to user-oriented design is that end-user practices and needs should inform the solutions being built. The understanding of the problem(s) end-users face, and potential solutions to these

should thus evolve as the process is carried out. For this to be possible, there must be some flexibility in the project to (re)define requirements based on issues and needs that are discovered along the process. On the other hand, several of the experienced ISG implementers explain that too flexible scopes could end up with "scope creeps" where emerging requirements make the project unmanageable. A major challenge for the ISGs is thus to balance flexibility versus predictability in the scope and requirements for the projects. Flexibility is needed for user-oriented processes where problems and solutions are explored as the process evolves, while predictability in terms of what the client organization will get out of the process, and the person-hours and workload it implies for the ISG is important. In many projects, most or all the requirements are defined and agreed upon in detail prior to any design and construction is initiated, and before any end-users are exposed to prototypes or even consulted regarding their practices and needs. A Mozambique ISG implementer reflects on a project: *"all requirements were defined from the outset – we had no room for requirements to emerge during the process"*. At the other end of the spectrum, some ISGs have worked with projects that are very flexible, as explained by the Malawi ISG lead: *"Sometimes they [the user organization] don't know what they want, but they know that they want something … this gives us more room to negotiate how the process should look like, and for the solution to emerge over time"*. The concern in the latter situation is that the flexibility might as well be misused by the managers in the user organization to *"constantly change and expand the scope"* (India ISG implementer) of the project. Flexibility might end up being exploited by project managers rather than providing room for problems and solutions to be explored and emerge over time through end-user interactions.

Some of the ISGs explain their strategies for dealing with this. The Mozambique ISG lead tells how they typically start with a rather strict and defined scope, but as the project moves along and needs that can be translated into useful features in DHIS2 are identified, he works to sell these ideas in to the user organization and expand the project scope "bits by bits". As he articulates, *"it's up to [him/them] to push to expand the scope based on opportunities along the way"*. The Malawi ISG lead explains that during negotiation of the project scope and their mandate, he attempts to define *"some pockets for refinement of requirements"* along the process.

4.2.2 "Convincing" user organizations of the need for end-user inquiries

A challenge reported as common by the ISGs is that the user organization's IT project managers do not appreciate the relevance and importance of end-user-oriented activities in the implementation process. And more so, what it will involve in practice. As explained by the lead implementer in the Mozambique ISG, in many of their projects, *"the agreement with [the clients] often do not allow us to go to the field"*, or the budget limits user interaction activities. She explains that they sometimes finance field trips themselves, and hope that the findings will feed into new innovations that can be negotiated into the project at a later point. The project we were involved with in India provides an illustrative example. We requested the client managers for access to visit health facilities to observe and interact with end-users to better understand their practices, challenges, and needs. We wanted to use this information as basis when working to address usability challenges reported by the user organization. Our request did, however, meet significant resistance from the client. One of the IT project managers of the user organization explained his hesitation: *"I feel it is more important that the [team of researchers and India ISG] support on the technical part - solving the problem, rather than understanding more problems"*. At the time of our inquiry, the project also suffered from many technical challenges, e.g., related to server performance. From a purely technical viewpoint their concern makes perfect sense. Yet, it illustrates a lack of knowledge – and sufficient clarification from our side on the relevance of working with end-users when addressing challenges related to usability.

4.2.3 The gravity of the generic features and adaption capabilities

To win contracts for projects, it is imperative for the ISGs to be competitive in terms of project costs. The ISGs aim to limit costs by leaning on the generic features of DHIS2, for which development and maintenance costs are taken care of by the core team in Oslo. This means that many needs and

opportunities for innovation are deemed too costly to implement in the solution, if not possible to support with readily available generic features. An experienced implementer illustrates the challenge through an example from a prior project: *"They had contracted an interaction design agency to design the app without thinking about DHIS2. They had many great thoughts and ideas, but which was based on having blank sheets. In reality, we had to take tracker [DHIS2 generic module] as a starting point […] in the end; we didn't use much of the design made by the interaction designers."*

More so, what is striking is how the generic features of DHIS2 play an important role in the negotiation of the project, the requirements gathering process, and the design and construction activity. In all of these activities, the focus shifts between what is needed in the particular context, and what is possible with the generic features and adaption capabilities of DHIS2 within the given budget and expectations defined in the project configuration. For experienced implementers, their knowledge of the features of DHIS2 forms a lens throughout the project, which seems to direct their attention towards aspects readily configurable, while drawing attention away from what is not. This is why the requirements gathering process in many cases only orients around data input and output needs – this is what can easily be configured using the configuration facilities. As articulated by a Ugandan implementer, through the whole process, they *"always think in terms of DHIS2 features and how to map the requirements to these"*. Not only does this affect what the ISGs look for in the requirements gathering process, and what is built during the design and construction phase, but also how projects are configured in terms of the process, and goals of the projects.

The ability to develop custom apps significantly extends the space for design and innovation of functionality and user interfaces. This is, however, costly in terms of development and maintenance and must hence be an explicit part of the project configuration, either upfront, or negotiated as the process moves along. This is not straightforward as DHIS2 often is sold in as a ready-to-use solution. Many ISGs tend to avoid custom app development altogether, as articulated by an implementer: *"We have developed an eye for what is for us and what is not. This makes us avoid getting into projects beyond what [the generic] DHIS2 can handle"*. Being more experienced in app development and negotiate custom development as part of the projects may expand the possibilities that the ISGs see in the given implementation, and hence also expand the space for user-oriented design and innovation. A Tanzania developer reflects on this: *"building apps force us to look at other aspects, such as the kinds of layouts and functionality that best will support the user"*.

5 Discussion and Conclusion

We started out with the following two research questions: 1) What are conditions that affect the potential for user-oriented design in the context of generic enterprise software implementation? And 2) what implications do these conditions have for vendors who want to promote user-orientation during implementation of their solutions?

We will first address the first question by articulating and discussing three conditions we see as prominent in our analysis before addressing the second by discussing their implications for vendors.

5.1 Three conditions for user-oriented design

Based on our examination of the implementation-level design process and the ISGs reflection on their practices and challenges, we define three conditions we see as fundamental to the potential for user-oriented design. We argue that the conditions represent what prior literature refers to as *boundary conditions* of the context of design (Zahlsen et al., 2020) in generic enterprise software implementations. The conditions are summarized in Table 1, and discussed below.

First, how projects are configured in terms of their scope, structure, and mandates affect the potential for user-oriented design. We see several examples in the analysis of challenges related to this condition, including the issue of balancing flexibility and predictability, and that of convincing the user organization of the relevance of user-oriented design. Many of the implementation projects have

the primary aim of replacing existing paper-based systems. Within such a scope, there might not be much flexibility to explore and address challenges beyond that of what data is to be collected and how this should be presented. We see similar examples in other research, for instance, reporting how 'the contract' strongly affects the possibility to address usability problems, and if, how and when users are involved during implementation-level design (Martin et al., 2007).

Condition	Description
The project configuration	The scope and structure of the project affects the relevance of and possibility to conduct user-oriented design
The implementation-level design practices of the ISG	If and how the practices (i.e., the "usual way of doing things" (Schatzki, 2019)) of the ISG is geared towards advocating, negotiating, and conducting user-oriented design.
The features and adaption capabilities of the generic software solution	The features and adaption capabilities of the generic software shape both the process and the product of implementation-level design

Table 1. Three conditions affecting the potential for user-oriented design during generic enterprise software implementation

Second, we see significant variation in the motivation for and competence in conducting user-oriented design in the ISGs. While some see themselves as "fighting the battle on behalf of the users", others prefer to avoid interaction with end-users during the process. This is possibly the most discussed obstacle to user-orientation in existing literature where low "usability maturity" of the development organization has been pointed out as a frequent challenge (Ardito et al., 2014; Earthy, 1998; Svanæs & Gulliksen, 2008). However, our study points to the importance of not only being motivated and able to conduct user-oriented design, but to negotiate it into the project configuration. We see some examples of how experienced ISGs are able to negotiate for flexibility to incorporate solutions to end-user challenges, even within rather strict project scopes.

Finally, the features and adaption capabilities of the generic software represent a powerful condition in our case. As discussed in existing literature, it largely determines what can be built within the financial bounds of the implementation project (Martin et al., 2007; Mousavidin & Silva, 2017; Sommerville, 2008). Beyond what is discussed in the literature, our findings indicate that the (limited) flexibility of the software not only affects what is built during the design and construction phase. Rather the 'gravity' of the generic features and adaption capabilities of the generic solution shapes how projects are configured in terms of scope and structure, and it acts as a lens during the requirements gathering process. As a lens, the features and adaption capabilities bring attention to the aspects that are supported and can be configured in the solution, while directing attention away from the aspects that cannot. If the adaption capabilities, as in the case of DHIS2, primarily orients around what data can be reported and how it is presented, this inevitably will be the major focus of the design process, leaving other aspects such as novel functionality and user interfaces in the dark. Aspects of the context of use and end-user needs that go beyond what is readily available might be deemed too costly to implement, or even overlooked as the software frames what to look for.

Prior literature discusses how the implementation-level design process is mainly about changing the organization according to the features of the generic software (Kallinikos, 2004; Martin et al., 2007; Vilpola, 2008). A more accurate description based on our findings is that the features and adaption capabilities shape the kind of design and innovation that takes place in the implementation-level design process. It directs attention towards practices, needs, and challenges within the specific user organizations that can easily be addressed with generic features, and leaves other aspects in the dark. It thus enables certain types of design and innovation, while constraining others. Where the generic solution directs focus seems to be manifested in the practices of the ISGs, how projects are configured, and the focus of requirements gathering.

5.2 Implications for vendors

We now address our second research question by discussing the implications of the three identified conditions for vendors who work to promote user-orientation during implementation of their solutions. Literature report that vendors increasingly work to support and promote design and innovation beyond their own boundaries (Foerderer et al., 2019; Wareham et al., 2014). For instance, SAP appears to invest significant resources in supporting and promoting design and innovation based on the specific needs of user organizations. Their book 'Design Thinking with SAP', and a plethora of resources directly aim to cultivate user-oriented design practices among their partner organizations (the equivalent to the ISGs in our case) during implementation-level design (*SAP Fiori*, 2020). In their words, the aim is to bring implementation-level design with SAP from 'digitization'- to 'digitalization' projects (Prause, 2020). We thus argue that these implications are relevant to vendors of generic enterprise software beyond DHIS2, and other participants within such 'ecosystems' (Dittrich, 2014; Foerderer et al., 2019; Rickmann et al., 2014) or 'design networks' (Koch, 2007).

5.2.1 Implications for Capacity building

To support and promote user-oriented design, the simple advice of "involve the end-users" as, for instance, seen in the Principles for Digital Development (digitalprinciples.org, 2019), and promoting generic methodologies such as User-Centered Design is not sufficient. Rather, the methods and approaches promoted must be apt for integration into the existing practices of the ISGs. The conditions affecting the potential for various forms of user-orientation must be considered in this work. In our project, our studies of how implementation-level design of DHIS2 unfold provide a fruitful basis for developing methods and guidelines that are mindful of the actual context of where they will be used.

One aspect of this, stressed by prior literature, is building motivation and competence to conduct user-oriented design (Ardito et al., 2014). However, in our analysis, we see that the ability to *advocate* and *negotiate* user-oriented design as part of the project configuration is as relevant. In our analysis, we see some interesting examples of strategies employed by some of the representatives of ISGs. For instance, the lead of the Mozambique ISG seems to possess valuable skills in negotiating for user-oriented innovation to emerge, even within inflexible project configurations. Vendors and researchers alike should seek to learn from such experiences and skills to build capacity for others to follow. As projects and ISG practices differ, promoting one method or process to fit all would be of limited value. In our project, we have initiated the development of a design method toolkit, taking into consideration different types of project configurations, and the specific features and adaption capabilities of DHIS2. This will provide ISGs with user-oriented methods that are realistic to integrate into their projects and sensitive to the design flexibility they face with DHIS2. The toolkit aims to build capacity both for conducting and negotiating user-oriented design in implementation projects.

5.2.2 Implications for Software Design - building the 'right' design space

We see that the features and adaption capabilities of the generic solution largely affect the focus and outcome of the implementation-level design process. This means that the vendor, through the features and adaption capabilities of the generic software solution, shapes what is to be of focus, and what can be built during implementation-level design (Bertram et al., 2012; Mousavidin & Silva, 2017). If limited and rigid, the configuration facilities may constrain the innovative capacity of implementation-level design, reducing the process to a standardized 'set-up and install' procedure. This could have dire consequences for design and innovation, and brings resemblance to cautions made by Kallinikos (2004) regarding the effects of rigid IT systems:

Coping with urgent and ambiguous situations often presupposes the ability of responding innovatively to these situations. Such an ability in turn is inextricably bound up with the capacity of reading/framing such situations properly. Rigidly dissociated from framing, action loses its intentional

component and tends to degenerate to mindless procedure of execution that may have devastating consequences (Kallinikos, 2004, p. 23)

Inflexible generic enterprise solutions may as such impede valuable IT innovation that could have emerged based on particular user needs within the organization. Implementation-level design is reduced to a 'mindless procedure of execution' rather than serving as an engine for user-oriented design and innovation. On the other hand, greater flexibility may introduce development and maintenance costs for the individual user organization. If costs are too high, as seen related to custom app development in our case, they may not utilize this flexibility. This represents a challenge but also an opportunity. Given the 'right' features and adaption capabilities, generic solutions can be a fruitful enabler of design and innovation and even be designed to direct attention to aspects of importance to secure usability and relevance for end-users. Platform architectures that give a basis for custom app development seem to be relevant regarding this (Farhoomand, 2007; Foerderer et al., 2019; Roland et al., 2017). However, means of providing flexibility while keeping costs of utilizing it minimal must be found. In collaboration with the DHIS2 core team, we are currently exploring resources that may reduce the efforts needed to develop custom apps. Measures that we explore include user interface libraries, and web components that support designers in assembling apps faster, and which leaves the costs of maintaining the components in the hands of the core team. The ideal result is a space for design that is *generative* (Bygstad, 2017; Msiska & Nielsen, 2017), yet considered sufficiently 'cheap' to utilize. Overall, the aim is to offer a 'design infrastructure' of software features that can be configured and extended to drive and support user-oriented design and innovation. Technical flexibility is seen in relation to the method toolkit and other resources building capacity and giving support to the process of implementation-level design.

5.3 Contributions and Concluding Remarks

In this paper, we have explored conditions for user-oriented design during implementation of generic enterprise software solutions. The contributions of the paper lie in a) our empirical insights into an increasingly relevant yet little explored context for designing IT, and b) our conceptualization and discussion of three conditions with implications for vendors who want to support and promote user-oriented design. We find many of the same challenges underlying our three conditions in existing studies, including studies focusing on bespoke software projects. It is possible to argue that the three conditions we identify are general to any IT project, regardless of being based on a generic enterprise software solution, or if building solutions bespoke. Our findings may, as such, also be relevant to the stream of literature around boundary conditions for user-oriented design in general (Edwards et al., 2010; Zahlsen et al., 2020). Yet, the implementation of generic enterprise software differs from bespoke software development. A core project aim is to limit costs of custom development and maintenance by relying on generic features designed and maintained to be used across many user organizations. Our analysis shows how the "gravity" of the generic software solution pulls on the process of negotiating the project configuration, and the generic features and adaption capabilities act as a lens throughout the implementation-level design process. We argue that the conditions and their implications are relevant to researchers and practitioners engaged with the design of generic enterprise software (Bansler, 2021; Koch, 2007; Mousavidin & Silva, 2017; Pollock et al., 2007), and enterprise software ecosystems (Foerderer et al., 2019; Wareham et al., 2014).

Our study is limited to examining the practices and challenges related to user-oriented design during implementation within one software ecosystem. Studies focusing on the same aspects in other software ecosystems and implementation projects could be useful in elaborating and modifying the conditions and implications presented in this paper. Particularly, following ongoing implementation projects, or examining projects deemed as particularly successful could provide valuable findings.

To conclude, many prominent generic enterprise solutions have a rusty reputation of being difficult to use and constraining the flexibility for user organizations to design and innovate IT based on their specific needs (Berente et al., 2019; Kaipio et al., 2017). While our study identifies challenges that

partly concur with this picture, we also see great potential for generic enterprise software solutions as supporting (as opposed to constraining) user-oriented design and innovation. Our study points towards that vendors may get rid of the rusty reputation of their generic solutions by seeing the aim, not as to develop a ready-to-use solution. Instead, the aim could be seen as to provide resources for a 'design infrastructure' supporting efficient user-oriented design and innovation during implementation into specific user organizations.

References

Ardito, C., Buono, P., Caivano, D., Costabile, M. F., & Lanzilotti, R. (2014). Investigating and promoting UX practice in industry: An experimental study. *International Journal of Human-Computer Studies*, *72*(6), 542–551.

Ardito, C., Buono, P., Caivano, D., Costabile, M. F., Lanzilotti, R., & Dittrich, Y. (2014). Human-centered design in industry: Lessons from the trenches. *Computer*, *12*, 86–89.

Bansler, J. P. (2021). Challenges in user-driven optimization of EHR: A case study of a large Epic implementation in Denmark. *International Journal of Medical Informatics*, 104394.

Baxter, G., & Sommerville, I. (2011). Socio-technical systems: From design methods to systems engineering. *Interacting with Computers*, *23*(1), 4–17.

Berente, N., Lyytinen, K., Yoo, Y., & Maurer, C. (2019). Institutional logics and pluralistic responses to enterprise system implementation: A qualitative meta-analysis. *MIS Quarterly*, *43*(3).

Bertram, M., Schaarschmidt, M., & von Kortzfleisch, H. F. (2012). Customization of Product Software: Insight from an Extensive IS Literature Review. In *Shaping the Future of ICT Research. Methods and Approaches* (pp. 222–236). Springer.

Bygstad, B. (2017). Generative innovation: A comparison of lightweight and heavyweight IT. *Journal of Information Technology*, *32*(2), 180–193.

Damodaran, L. (1996). User involvement in the systems design process-a practical guide for users. *Behaviour & Information Technology*, *15*(6), 363–377.

digitalprinciples.org. (2019). *Principles for Digital Development*. Principles for Digital Development. https://digitalprinciples.org/

Dittrich, Y. (2014). Software engineering beyond the project–Sustaining software ecosystems. *Information and Software Technology*, *56*(11), 1436–1456.

Earthy, J. (1998). Usability maturity model: Human centredness scale. *INUSE Project Deliverable D*, *5*, 1–34.

Edwards, W. K., Newman, M. W., & Poole, E. S. (2010). The infrastructure problem in HCI. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 423–432.

Ellingsen, G., & Hertzum, M. (2019). User participation in the implementation of large-scale suite systems in healthcare. *Proceedings of the 7th International Conference on Infrastructures in Healthcare*, *4*(3). https://doi.org/10.18420/ihc2019_002

Farhoomand, A. (2007). Opening up of the software industry: The case of SAP. *Eighth World Congress on the Management of EBusiness (WCMeB 2007)*, 8–8.

Foerderer, J., Kude, T., Schuetz, S. W., & Heinzl, A. (2019). Knowledge boundaries in enterprise software platform development: Antecedents and consequences for platform governance. *Information Systems Journal*, *29*(1), 119–144.

Gay, G., & Hembrooke, H. (2004). *Activity-centered design: An ecological approach to designing smart tools and usable systems*. Mit Press.

Gizaw, A. A., Bygstad, B., & Nielsen, P. (2017). Open generification. *Information Systems Journal*, *27*(6), 619–642.

gov.uk. (2019). *Government Design Principles*. GOV.UK. https://www.gov.uk/guidance/government-design-principles

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, Å. (2003). Key principles for user-centred systems design. *Behaviour and Information Technology*, *22*(6), 397–409.

Hustad, E., Haddara, M., & Kalvenes, B. (2016). ERP and organizational misfits: An ERP customization journey. *Procedia Computer Science*, *100*, 429–439.

Kaipio, J., Lääveri, T., Hyppönen, H., Vainiomäki, S., Reponen, J., Kushniruk, A., Borycki, E., & Vänskä, J. (2017). Usability problems do not heal by themselves: National survey on physicians' experiences with EHRs in Finland. *International Journal of Medical Informatics*, *97*, 266–281.

Kallinikos, J. (2004). Deconstructing information packages: Organizational and behavioural implications of ERP systems. *Information Technology & People*, *17*(1), 8–30.

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *Mis Quarterly*, 67–93.

Koch, C. (2007). ERP-a moving target. *International Journal of Business Information Systems*, *2*(4), 426.

Li, M. (2019). An Approach to Addressing the Usability and Local Relevance of Generic Enterprise Software. *Selected Papers of the IRIS*, *10*, 1–15. https://aisel.aisnet.org/iris2019/3

Li, M., & Nielsen, P. (2019). *Making Usable Generic Software. A Matter of Global or Local Design?* 10th Scandinavian Conference on Information Systems (SCIS), Nokia, Finland.

Magnusson, J., Klingberg, J., Enquist, H., Oskarsson, B., Nilsson, A., & Gidlund, A. (2010). All Aboard: ERP Implementation as Participatory Design. *AMCIS*, 253.

Martin, Dave, Procter, R., Mariani, J., & Rouncefield, M. (2007). Working the contract. *Proceedings of the 19th Australasian Conference on Computer-Human Interaction: Entertaining User Interfaces*, 241–248.

Martin, David, Mariani, J., & Rouncefield, M. (2007). Managing integration work in an NHS electronic patient record (EPR) project. *Health Informatics Journal*, *13*(1), 47–56.

Mathiassen, L., & Nielsen, P. A. (2008). Engaged scholarship in IS research. *Scandinavian Journal of Information Systems*, *20*(2), 1.

Mousavidin, E., & Silva, L. (2017). Theorizing the configuration of modifiable off-the-shelf software. *Information Technology & People*, *30*(4), 887–909.

Msiska, B., & Nielsen, P. (2017). Innovation in the fringes of software ecosystems: The role of socio-technical generativity. *Information Technology for Development*, 1–24.

Mumford, E. (2006). The story of socio-technical design: Reflections on its successes, failures and potential. *Information Systems Journal*, *16*(4), 317–342.

Myers, M. D. (2019). *Qualitative research in business and management*. Sage.

Nielsen, J. (1994). *Usability engineering*. Elsevier.

Norman, D. (2013). *The design of everyday things: Revised and expanded edition*. Constellation.

Norman, D. A. (2005). Human-centered design considered harmful. *Interactions*, *12*(4), 14–19.

Pollock, N., Williams, R., & D'Adderio, L. (2007). Global software and its provenance: Generification work in the production of organizational software packages. *Social Studies of Science*, *37*(2), 254–280.

Prause, J. (2020, November 18). *Digitization vs Digitalization*. SAP Insights. https://insights.sap.com/digitization-vs-digitalization/

Pries-Heje, L., & Dittrich, Y. (2009). ERP implementation as design: Looking at participatory design for means to facilitate knowledge integration. *Scandinavian Journal of Information Systems*, *21*(2), 4.

Rickmann, T., Wenzel, S., & Fischbach, K. (2014). *Software ecosystem orchestration: The perspective of complementors*.

Roland, L. K., Sanner, T. A., Sæbø, J. I., & Monteiro, E. (2017). P for Platform: Architectures of large-scale participatory design. *Scandinavian Journal of Information Systems*, *29*(2).

Rosson, M. B., & Carroll, J. M. (2002). *Usability engineering: Scenario-based development of human-computer interaction*. Morgan Kaufmann.

SAP Fiori. (2020). SAP. https://www.sap.com/products/fiori.html

Schatzki, T. R. (2019). *Social Change in a Material World*. Routledge.

Sestoft, P., & Vaucouleur, S. (2008). Technologies for evolvable software products: The conflict between customizations and evolution. In *Advances in Software Engineering* (pp. 216–253). Springer.

Sia, S. K., & Soh, C. (2007). An assessment of package–organisation misalignment: Institutional and ontological structures. *European Journal of Information Systems*, *16*(5), 568–583.

Simonsen, J., & Robertson, T. (2012). *Routledge international handbook of participatory design*. Routledge.

Soh, C., & Sia, S. K. (2008). The challenges of implementing" vanilla" versions of enterprise systems. *MIS Quarterly Executive*, *4*(3), 6.

Sommerville, I. (2008). Construction by Configuration: Challenges for Software Engineering Research and Practice. *19th Australian Conference on Software Engineering (Aswec 2008)*, 3–12. https://doi.org/10.1109/ASWEC.2008.4483184

Strong, D. M., & Volkoff, O. (2010). Understanding Organization—Enterprise system fit: A path to theorizing the information technology artifact. *MIS Quarterly*, 731–756.

Svanæs, D., & Gulliksen, J. (2008). Understanding the context of design: Towards tactical user centered design. *Proceedings of the 5th Nordic Conference on Human-Computer Interaction: Building Bridges*, 353–362.

Sykes, T. A., & Venkatesh, V. (2017). Explaining post-implementation employee system use and job performance: Impacts of the content and source of social network ties. *MIS Quarterly*, *41*(3), 917–936.

Tavory, I., & Timmermans, S. (2019). Abductive analysis and grounded theory. *The SAGE Handbook of Current Developments in Grounded Theory*, 532–546.

Van de Ven, A. H. (2007). *Engaged Scholarship: A Guide for Organizational and Social Research*. OUP Oxford.

Vilpola, I. H. (2008). A method for improving ERP implementation success by the principles and process of user-centred design. *Enterprise Information Systems*, *2*(1), 47–76.

Wareham, J., Fox, P. B., & Cano Giner, J. L. (2014). Technology ecosystem governance. *Organization Science*, *25*(4), 1195–1215.

Zahlsen, Ø. K., Svanæs, D., Faxvaag, A., & Dahl, Y. (2020). Understanding the Impact of Boundary Conditions on Participatory Activities. *Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences, Shaping Society*, 1–11.

III

IV