Building Boundary Resources for Enterprise Software Platforms

Factors Affecting the Fit of a Software Enterprise Platform's Boundary Resources in Implementation-level Contexts in India and Tanzania

Kristine Jevne Berge

Master Thesis submitted for the degree of
Master of Informatics: Programming and System Architecture
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring / 2020

Building Boundary Resources for Enterprise Software Platforms

Factors Affecting the Fit of a Software Enterprise Platform's Boundary Resources in Implementation-level Contexts in India and Tanzania

Kristine Jevne Berge

2020

Abstract

This thesis investigates the introduction of an enterprise software platforms new boundary resources in an implementation-level context. More specifically, it focuses on how these new boundary resources, aimed at strengthening the development of custom applications, can fit in two organizations specialized in implementing the enterprise software DHIS2 and their practices in app development, and what the challenges are.

Many enterprise software vendors have utilized the *platform strategy* and “opened” their technology to third parties. The platforms layered architecture and boundary resources facilitate for third party complementors, and thus to provide the enterprise software with more flexibility to address the challenges of misfits between the enterprise software and use-specific context.

In a two-year case study, three contexts were investigated; the *core team,* i.e. the enterprise vendor in Oslo, and two implementation-specialist groups located in Noida, India, and in Dar es Salaam, Tanzania. The core team provided insight into the resources they made and why, and the implementation-specialist groups gave insight into their implementation practices and challenges, with a particular focus on app development.

The empirical findings of this thesis show custom app development is found to be a challenging and time-intensive task for the two implementation-specialist groups, and that the two groups have developed many resources for themselves to ease some of the challenges. New boundary resources made by the core team is aimed at reducing some of this cost and make app development an easier task. However, there are found to be several challenges related to the fit of boundary resources. The analysis shows that the fit of boundary resources needs to be seen in a socio-technical perspective, as the fit must be seen in relation to technologies used in implementations and the implementation-level contexts’ capacity and existing practices. To understand and elaborate this, the concepts of *local design infrastructures* and *global design infrastructure* is been proposed to describe the dynamics between the implementation-specialist groups and the core team, and their various co-existing structures, resources, practices, and activities.

Keywords: Enterprise software, enterprise software platform, boundary resources, self-sourcing, design infrastructures, global design infrastructures, local design infrastructures.

Acknowledgments

I am extremely grateful to my supervisor Magnus Li for his unwavering guidance and support throughout the last two years. I also want to thank you for the many interesting discussions, constructive criticism, and advice that have made this thesis possible. I am thankful to the HISP group at the University of Oslo, for all their help and support. I would also like to express my deepest appreciation to my fellow researchers Rebekka, Elisabeth, Fredrik, and Terje, for our many interesting discussions, helpful advice, and most importantly; your companionship during our long stays in India. I would like to extend my sincere thanks to HISP India, HISP Tanzania, and the DHIS2 Core development team that has participated in the project and made considerable contributions. A special thanks goes to HISP India for including us and allowing us to stay for such a long time. I also gratefully acknowledge the assistance of the DHIS2 Design Lab for the many interesting discussions and insights.

Finally, I would like to thank family, friends, Emilie, and Sindre for unparalleled support and encouragement throughout the process.

Kristine Jevne Berge

University of Oslo

June 2020

Table of Content

List of Figures

List of Tables

1. Introduction

Based on qualitative research conducted over the past 2 years, this thesis focuses on resources and tools developed to support the implementation of a generic "packaged" enterprise software that has a broad range of uses.

Enterprise software is software used to satisfy needs of an organization. *Generic* "packaged" enterprise software solutions have become increasingly more popular to use within organizations, as they are often cheaper than custom solutions (Dean et al., 2002; Soh & Sia, 2008). Custom software is designed to fit a specific organization's established practices and needs, whereas generic enterprise software aims at serving a multitude of organizations. Thus, generic enterprise software cannot be too specific to a given context, as it can then become irrelevant for other organizations (Berente et al., 2019; Li, 2019a). Research has shown that "pushing organizations to fit into standardized software packages is no easy task; it has led to many misfit failures" (Gizaw et al., 2017, p. 622), and that there is often a 'misfit', or 'misalignment', between the generic enterprise software itself and the intended user organizations' practices (Sia & Soh, 2007). This is a result of the generic enterprise software not being designed for the specific context and the end-users are often unwilling to change their work practices to fit the software. Generic enterprise software packages are therefor often too generic to be used "out of the box", and many researchers have argued that generic enterprise software must to be customized, or adapted, to become a better fit for the specific context in which it is to be used (Bansler & Havn, 1994; Gizaw et al., 2017). Some have argued that this adaptation, through design and innovation, should be possible to do when the software is implemented into specific user organizations (Li & Nielsen, 2019).

Many vendors of generic enterprise software have "opened" their technology to allow for third parties to contribute and create extensions to the software, a strategy beneficial to facilitate for and promote innovation (Foerderer et al., 2019). This is known as the *platform strategy* where the generic enterprise vendor implements a *platform architecture* to enable third parties to complement the software by developing custom applications that extend the generic enterprise software's functionality. The platform architecture is coupled with a range of resources provided by the enterprise software vendors to support and control the efforts of the complementing parties (Ghazawneh & Henfridsson, 2013). These resources are described as *boundary resources*. Even though the platform strategy is increasingly used by enterprise software vendors, there is limited research on how to support design and innovation during the implementation of the enterprise software in specific user organizations (Li & Nielsen, 2019).

In many countries, generic enterprise software is an important component in Health Information Software (HIS). The generic enterprise software package DHIS2 has been implemented and is used in over 60 countries, and it is the largest Health Management Information System (HMIS) in the world. DHIS2 was an initiative by the global movement Health Information Systems Program (HISP). The HISP network consists of several organizations, which are spread across the globe, working on strengthening health through HIS. The various HISP organizations in the network are specialized in implementing DHIS2 and can thus be referred to as *implementation-specialist groups.* The implementation-specialist groups implement and maintain instances of DHIS2 on behalf of various governments and organizations. The context in which the implementation-specialist groups implement DHIS2 will be referred to as the implementation-level context.

The generic enterprise software DHIS2 has provided a platform architecture and a variety of resources to make it adaptable and become a better fit for the use-specific context. There are several approaches to adapting DHIS2 enterprise software. The main approach is through the *configuration* of the software, which is achieved through changing and setting predefined options that the software comes with (Dittrich, 2014; Li, 2019b). As DHIS2 is increasingly being used for other domains besides health, configuration is not always a viable approach. Therefore, creating custom applications and customization of the software's core code is not uncommon. Custom applications are created as extensions to the software. The challenge with custom applications is that they are associated with significant financial costs related to the time and competence needed for development and maintenance, compared to the cost of configuration. At the same time extensions provide high flexibility in design, as the extension can be made to be exactly to what is needed for the specific context in which the enterprise software is implemented (Roland et al., 2017).

This thesis is part of the broader ongoing Action Research project which the HISP research group is conducting. The HISP research aims to gain an understanding of how to better "support local management of health care delivery and information flows in selected health facilities, districts, and provinces, and its further spread within and across developing countries" (*Health Information Systems Programme (HISP)*, n.d.). Within the HISP research project, the research group "DHIS2 Design lab" focuses on the DHIS2's software and the surrounding resources, people, and practices, which is referred to as the software's *design infrastructure*. The lab aims to "strengthening the implementation-level design and innovation capacity of the design infrastructure so that it better supports designers in locally building the software that is right for particular contexts of use" (*DHIS2 Design Lab*, n.d.). In other words, the lab aims at understanding how tools and resources can facilitate DHIS2 implementations in various implementation-level contexts, and how DHIS2 can support a variety of needs and a diverse audience.

As a part of the DHIS2 Design labs research, the focus of this study will be on how the vendor of DHIS2 build resources to support the development of custom applications, and how these can be used on the level of implementation. In the empirical case, we follow the vendor's development team, and two implementation-specialist groups; HISP India and HISP Tanzania.

1.1 Motivation

It has been argued that more research is needed on processes in which the software is implemented into specific user organizations to understand which parts of generic enterprise software that should be "customizable, and how it can be made easy and efficient, without interfering with the updatability" (Li & Nielsen, 2019) of the enterprise software. There has also been a call for more research on the technology itself in the research of the implementation of enterprise software. Berente et al. (2019) argue that "most of the attention in these studies focuses on organizational and social dynamics while the role and impact of technology goes largely untheorized" (p. 898). Research has largely focused on economic aspects of the platform strategy, rather than on innovation dynamics (de Reuver et al., 2018). De Reuver et al. (2018) argues that this "strand of research does not facilitate an opening of the technological black box necessary to understand platform generativity and other innovation dynamics" (p. 126). Little is known about how to make good boundary resources for implementation-specialist groups, which are organizations that are neither vendors nor are they the intended end-user organization. To contribute to filling this gap, I will look at boundary resources that are aimed at supporting the development of custom applications in an implementation-level context and investigate how and if these resources can fit into this context.

1.2 Research Question

The research aims to address how problems at the implementation level, such as the high cost of developing custom applications, can be approached through the involvement of the software enterprise platform vendor. The intent is to get a better understanding of how the software enterprise vendor can provide boundary resources to support implementations of custom applications, and how these resources fit with the existing application development practices in the implementation-level context. The research question is as follows;

What affects a platform's boundary resources fit in an implementation-level context?

To answer the research question the objective is to

1) understand the current practices related to how applications currently are developed for DHIS2 at implementation-level, which provides a basis for the following objective,
2) investigate the challenges related to implementation-level app development,
3) to analyze (if or) how the new resources developed by the DHIS2 vendor, aimed at strengthening app development can fit into the process of implementation-level design, and
4) discuss how this relates to existing research literature.

The empirical data will be based on research from three different contexts; two DHIS2 implementation-specialist groups 1) HISP India, and 2) HISP Tanzania, and 3) the core development team of DHIS2. The implementation-specialist groups provided insight into the practices related to the custom development of applications. From the core development team, insight was gained on how they developed resources aimed at strengthening and promoting the implementation-level development of custom applications.

1.3 Chapter Summary

Chapter 2: Background

Provides a general background for the project and introduces the HISP project and the DHIS2 software. It will also introduce two implementation-specialist groups HISP India and HISP Tanzania, as well as one of HIPS India's clients Alpha Consulting. Additionally, present some technical background on app development frameworks and the DHIS2 newly developed app resources.

Chapter 3: Related literature

Introduces literature relevant to understanding 1) Enterprise software platform, and the related 2) platform architecture and 3) resources surrounding the platform, as well as some of the challenges related to these elements. Further, it elaborates on theoretical concepts that will be used to analyze and understand the findings, which are related to enterprise software platforms as design infrastructures, and the related processes of generic-level design and implementation-level design that takes place within the design infrastructure.

Chapter 4: Methodology

Describes and explains the research methodology, methods, and techniques used in the research, as well as how the data was collected, as well as presenting the research context and challenges.

Chapter 5: Findings

Relate to the two first objectives of this thesis, with five main parts: 1) presenting the existing resources DHIS2 provide to support implementation-level design, 2) the current practices of app development in these implementation-level contexts, 3) the locally-developed resources to support app development practices, 4) challenges with app development, and 5) the new boundary resources provided by the DHIS2 core team aimed at strengthening app development at implementation-level.

Chapter 6: Analysis

Relates to the third objective of the research. Presents an analysis of how the new boundary resources can fit with the existing practices and challenges within the implementation-specialist groups, by looking at the benefits of the new boundary resources, and the challenges they pose in this context.

Chapter 7: Discussion

Discussion of how the findings and results of the analysis relate to other literature.

Chapter 8: Conclusion

Summary of the main findings and reflects on future work.

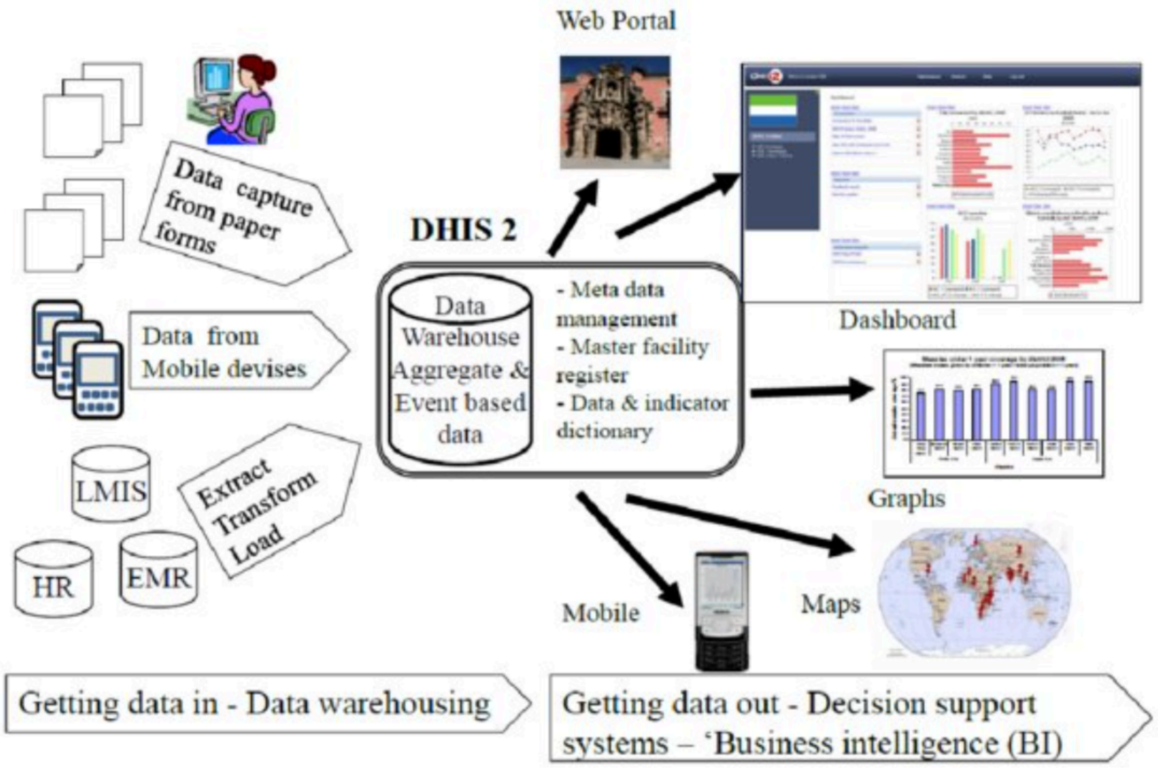
2. Background

This chapter will first provide a brief background on the HISP project and the DHIS2 software. Then, the DHIS2 Design Lab will be presented, as this thesis is a part of the research within the lab. Next, the implementation-specialist group HISP India will be presented, followed by a brief introduction to one of their clients, Alpha Consulting, whom we got to visit and interview during our research. After this, another implementation-specialist group, HISP Tanzania will be introduced. Following this, some frequently used frameworks in application development will be presented, as this gives us some technical insight into how applications are made, which will be useful to know when I later investigate how the boundary resources fit with the implementation-level contexts. Lastly, a brief introduction of DHIS2's new resources will be provided.

2.1 HISP Project and DHIS2

Health Information System Program (HISP) defines itself as a global movement aimed at improving health through strengthening Health Information Systems in Developing countries (*Health Information Systems Programme (HISP)*, n.d.). HISP has implementation-specialist groups across the world. All the implementation-specialist groups aim at strengthening HIS, working together with governments at different levels, as well as global organizations, such as WHO, UNICEF, and others. At the core of HISP is DHIS2 (District Health Information System 2), a free and open-source HIS software package. This software is currently used in about 72 countries (*DHIS2 Front Page*, n.d.), and HISP contributes through capacity building and supporting the implementation of DHIS2. The HISP organization located at the University of Oslo consists of two groups; researchers and the "core team", that drive the design, development, and maintenance of the DHIS2 software. Researchers range from master students to professors, researching the different aspects of DHIS2 in many different countries. The DHIS2 system's structure can be divided into several layers, as illustrated in Figure 2 in Chapter 3. These layers consist of a core, bundled applications, and custom applications (Roland et. al., 2017, pp. 25-26). The core team consists of thirty fulltime employees, who are mainly developers.

The DHIS2 generally serves as a *data warehouse* that can store and access data from various sources. The data can come from an integration with other systems, such as other HIS software, web-applications, or mobile reporting. Furthermore, DHIS2 comes with a range of built-in bundled applications as well as the possibility to integrate with custom applications. These applications can be used to analyze and visualize the data. The architecture is illustrated in Figure 1. The core team are also responsible for developing, designing, and maintaining the bundled applications.



Web Portal
Data capture from paper forms
DHIS 2
Data Warehouse Aggregate & Event based data
- Meta data management
- Master facility register
- Data & indicator dictionary
Dashboard
Data from Mobile devises
Graphs
LMIS
Extract Transform Load
HR
EMR
Mobile
Maps
Getting data in - Data warehousing
Getting data out - Decision support systems – 'Business intelligence (BI)

Figure 1: DHIS2 Architecture (Braa & Sahay, 2017).

2.2 The DHIS2 Design Lab

The DHIS2 Design Lab is a part of the HISP research project at the University of Oslo. The DHIS2 Design Lab aims to explore implementation-level design, and how usability can be addressed at this level. The lab is led by a Ph.D. candidate and consists of a group of master students. The idea with the lab is to explore various aspects of how the enterprise software is designed during implementation and aims at strengthening this process and how resources can facilitate innovation. My project in connection to the design lab is focused on the newly developed resources aimed at strengthening the development of extensions, or applications, for DHIS2.

2.3 HISP India

HISP India is one of the implementation-specialist groups in the global HISP network, whose office is located in Noida, Uttar Pradesh, India. The company consists of about 40 employees (HISP India, n.d.-a). The employees can be divided into two groups; technical and public health. Currently (02.04.2020) six employees in the technical team deal with software development, server administration, and testing. The public health group, called *implementers*, consists of 19 employees who work with, inter alia, requirements analysis, design, capacity building, system implementation, and documentation (HISP India, n.d.-a). HISP India is working on several projects across the globe. Their projects range from the implementation of national or regional HMIS to vertical health programs, such as HIV monitoring or Antimicrobial Resistance (AMR) monitoring, to other projects that are not related to health, such as tracking of youth unemployment (HISP India, n.d.-b). Their clients mainly vary between governments and NGO's. They mainly work with implementations of DHIS2, but also do integrations between DHIS2 and other systems. In addition to offering implementation and maintenance of DHIS2 instances, HISP India also offers academies, in collaboration with the University of Oslo, for DHIS2 users in the region (HISP India, n.d.-c).

2.4 Alpha Consulting

One of HISP India's largest contracts is with a non-governmental organization, here referred to as Alpha Consulting. Alpha Consulting was contracted by the government in the state Uttar Pradesh (UP) to implement a HIS in the state. Alpha Consulting contracted HISP India to deal with the technical implementation of the system, and Alpha Consulting would be responsible for training users and following up on the system. Alpha Consulting's offices are located in UP's capital, Lucknow, which is about 500 kilometers southwest from New Delhi. Uttar Pradesh is the largest state in India in terms of population, with roughly 200 million inhabitants. Implementing a HIS for this many people is therefore a quite comprehensive task.

2.5 HISP Tanzania

HISP Tanzania is another implementation-specialist group in the HISP network. The organization has had been involved in several implementations of DHIS2. In Tanzania, they have implemented several instances for use within health, but also within a wide range of other domains such as water, and agriculture, as well as on behalf of international organizations. They have also been projected to do implementations in other countries such as with Health Ministries of Kenya, Uganda, Malawi, Zimbabwe, Zambia, and Democratic Republic of Congo (HISP Tanzania, n.d.).

2.1 Application Development Frameworks

When developing applications in general, it is common to use an app development framework. There are currently three very popular frameworks/libraries for creating the frontend of web applications. These frameworks are React, Vue, and Angular (Martin, n.d.). These frameworks allow the developer to more efficiently create applications with JavaScript, HTML, and CSS, which are the essential building blocks in web programming. The different frameworks offer, in essence, the same functionality. However, the structure of which the programmer has to write code differs from the different frameworks. In other words, to be able to use all the frameworks, the developer has to learn how to use each and one of them. In addition to the frameworks, it is also common in web application development to utilize libraries with premade functionality, UI components, or both. Some libraries are supported by all three of the frameworks, however, many libraries are only supported by one of them. In other words, a library imported into a React application cannot necessarily be imported into an Angular application. The libraries utilized in the applications are often installed and imported into the application using a package manager.

2.2 DHIS2 Newly Developed App Resources

When the research for this thesis started, the new resources developed by the core team was in its initial stage. These resources are the DHIS2 Design System and the DHIS2 App platform. The Design System is a component library consisting of standalone components in DHIS2-style, as well as design principles and standards for how to use the components. The App platform is a unified architecture and build, which serves as a sort of a template when starting on a new application. The new resources will be discussed further in Chapter 5, which will also elaborate on how the resources evolved throughout the project.

3. Related Literature

The thesis aims to understand what affects platform boundary resources fit in an implementation-level context. To do so, other research contributions related to the thesis' research topic and theoretical concepts used in the analysis and discussion, will be presented. Firstly, the chapter will go through enterprise software and the challenges of providing flexibility to support various uses, while at the same time provide stability. I will then present how some enterprise software vendors have addressed the challenges by implementing a platform architecture, which allows other parties to complement the software to make it a better fit for the context-specific use. Following, the chapter will elaborate on the research done on platform architectures, both how the architecture itself is structured as well as the surrounding people and resources that facilitate and promote new uses for enterprise software platforms, while at the same time provide stability. Next, the concept of "Design Infrastructures" will be presented as it is a useful concept to describe all the activities, people, and technology that surrounds an enterprise software platform. After this, I will conceptualize two key processes related to the design of generic enterprise software, which happen at different "levels"; generic-level design and implementation-level design. These will be used to understand how resources are developed aimed at promoting and supporting customization and adaption of enterprise software. It will also help us understand the ramification of the resources in various contexts, and how these resources may fit with the practices of implementation-level design context.

3.1 Generic Enterprise Software

Generic enterprise software is, as mentioned in the introduction, software which is not designed for a specific user organization in mind. Organizations are increasingly relying on the use of enterprise software, rather than in-house development of custom software. This is mainly due to economic aspects, as the cost of creating and maintaining custom software is rising (Bansler & Havn, 1994). An enterprise software vendor can not facilitate or account for all the diverse requirements that various user organizations might need, as the requirements can be numerous and different organizations can have conflicting requirements (Gizaw et al., 2017; Pollock et al., 2007). Hence, there is often a misfit between enterprise software packages and the specific context, practices, and requirements of the user organization (Gizaw et al., 2017). However, to address the diverse and dynamic requirements, the enterprise software aims at providing flexibility to facilitate for adaption of the software in the local context, to become a better fit for the specific requirements. Since enterprise software facilitate for more local adaption and customization of the software, technology innovation is increasing as a distributed activity, taking place in networks and ecosystems (Gizaw et al., 2017). There has been a lot of research on approaches to customizing enterprise software (Bansler & Havn, 1994; Brehm et al., 2001; Domingos et al., 1997; MacLean et al., 1990), and it has been suggested that the enterprise software's architecture is one of the keys to supporting a large number of techniques to customizing the software (MacLean et al., 1990).

3.1.1 Generic Enterprise Software Platform

To address the challenge of misfits between generic enterprise software and the use-specific context, many vendors of enterprise software have decided to "open" their technology by allowing third parties to contribute to the software by creating *applications* that are connected to the software. For the vendor, the rationale behind this is to promote innovations and to support more uses of the software (Foerderer et al., 2019). This is known as the "platform strategy", which is increasingly adopted by enterprise software vendors (Foerderer et al., 2019). Research has mainly focused on two correlated and enabling factors of platforms that make it possible for third parties to contribute and develop complements for the platform. These two factors are the platforms layered architecture (Baldwin & Woodard, 2009; Tiwana, 2014), and the *boundary resources* and *boundary spanners*, i.e. the people, tools, and regulations, that facilitate the relationship between the platform vendor and the platform complementors (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013; Henfridsson & Bygstad, 2013).

3.1.2 Platform Architecture

The platform architecture is layered, as illustrated in Figure 2. The inner layer consists of a generic core that provides a generic UI and some core functionality. This layer has low flexibility in terms of what the user organization can change, but it has high flexibility in terms of being ready to use for a range of different purposes (Baldwin & Woodard, 2009; Tiwana, 2014). The outer layer consists of custom applications, or extensions, to the software.

The development of custom applications provides the possibility of high design flexibility without affecting the generic core and thus does not interfere with updatability.

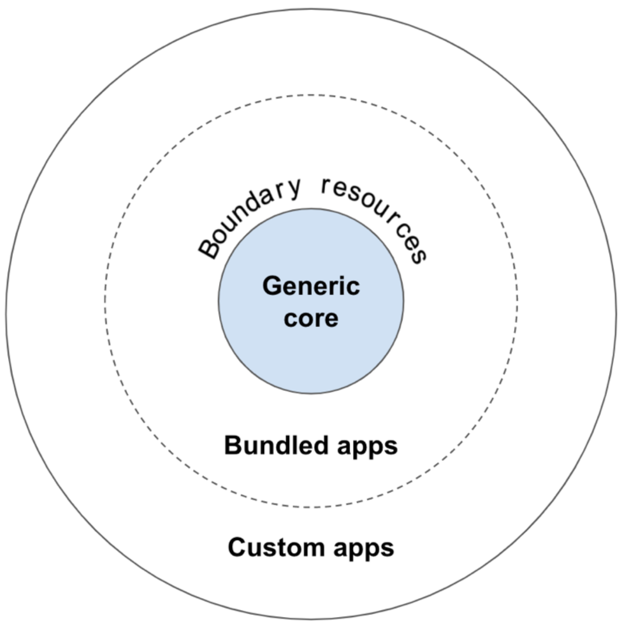


Figure 2: Platform architecture, based on Roland et al.'s figure (2017, p. 25)

The middle layer binds together the outer and inner layers. This is also the key to understanding how platform architecture paradoxically can provide both flexibility and stability (de Reuver et al., 2018; Tilson et al., 2010). The middle layer allows for implementing organizations to customize the software and other pre-made applications, through for example configurations. These premade applications can be referred to as "bundled applications" as they come together with the generic core (Roland et al., 2017). The middle layer also consists of boundary resources. Boundary resources are tools and regulations that facilitate the relationship between the platform vendor and third-party complementors, and allows the complementors to develop custom applications (Eaton et al., 2015). Platforms can provide a range of boundary resources, such as APIs or Software Development Kits (SDKs), which allows for third-party applications to be connected to the platform (Roland et al., 2017). These third-party applications do not intervene with the core's code but are created as separate modules, connected to the platform through the platform's boundary resources. In other words, the boundary resources enable others to extend the platform and provide new functionality and features. Hence, applications made by third parties who utilize the boundary resources the middle layer provides to do so.

By utilizing the architecture of platforms, the vendors of enterprise software can allow third parties to contribute and participate in the development, which can provide increased flexibility for the software's user organizations. The platform strategy also has the benefit of moving some of the development from within the platform vendor to independent platform complementors (Foerderer et al., 2019). The architecture supports a community of complementors who can creatively develop extensions and utilize the platform in unforeseen ways (Foerderer et al., 2019). Hence, the platform architecture has been argued to promote innovations (de Reuver et al., 2018; Gawer & Cusumano, 2014). The platforms layered architecture provides stabilization at one level, while at the same time flexibility by allowing complementors of the enterprise software platforms to extend the software for a variety of uses (Baldwin & Woodard, 2009). Thus, the platform architecture can create an "unprecedented scope for innovation on complimentary products, services, and technologies" (Gawer & Cusumano, 2014, p. 421). It is argued that competition among platform vendors is no longer revolving around "how to control the value chain but around attracting generative activities associated with a platform" (de Reuver et al., 2018, p. 124).

3.1.3 Boundary resources

Platform research has focused on a variety of different aspects related to platform management and governance (Foerderer et al., 2019; Wareham et al., 2014). Some have argued that the platform's boundary resources stability is an important property of software platforms (Bianco et al., 2014; Robbes & Lungu, 2011). Too big changes in the boundary resources has shown to have "trickle-down" effects on the complementors, as instability can lead to higher maintenance work, and thus leading to platform complementors deeming the software platform undesirable (Bianco et al., 2014; McKenzie, 2013). Others have argued that platforms pose a paradox, as the platform must be stable to attract complementors, while at the same time provide flexibility to allow for growth (Baldwin & Woodard, 2009; Tilson et al., 2010).

It has been claimed that tensions between platform vendors and complementors can arise due to competition between the two (Baldwin & Woodard, 2009). The boundary resources can thus be used to by the platform vendor control and regulate the platform to avoid that the complementors develop applications that compete with what the platform vendors provides or prevent the complementors from "cloning" the platform to compete directly (Baldwin & Woodard, 2009). Ghazawneh and Henfridsson (2013) have suggested that the platform vendors key to balancing generativity and control by the is through the platform's boundary resources. They argue that two processes are driving the boundary resource design; 1) resourcing expands the scope and enhances the diversity of the platform, and 2) securing is the process in which the platform vendor's control over the platform is increased (Ghazawneh & Henfridsson, 2013). They also define the concept of *self-resourcing*, which is the process in which third-party complementors develop new boundary resources "as a response to perceived limitations in existing boundary resources" (Ghazawneh & Henfridsson, 2013, p. 186), and explore how the platform vendors have address undesired and unanticipated self-resourcing by balancing resourcing and securing of their platform. Their contribution to understanding platform management through digital innovations, by exploring "boundary resources as digital technology with the capacity to trigger innovation driven by multiple and uncoordinated third-party developers" (Ghazawneh & Henfridsson, 2013, p. 187). It has been pointed out by other researchers that in Ghazawneh and Henfridsson's research, it is "not clear how boundary resources come into being and exactly how they evolve over time" (Eaton et al., 2015, p. 220).

It has also been emphasized that Ghazawneh and Henfridsson's study is taking the privileged perspective of the platform vendor, who largely designs and controls the boundary resources, leaving third-party complementors as "passive recipients of these boundary resources" (Eaton et al., 2015, p. 220). Eaton e. al. (2015) argue that third-party complementors are not merely passive receivers of boundary resources, but rather they are actively involved in processes of "distributed tuning", where the resources are shaped and reshaped by complementors by the platform vendors. Through the distributed tuning of the boundary resources the actors, both the platform vendor and the complementors, aim at gaining control to achieve their respective, differing, goals (Eaton et al., 2015).

Some research has focused on how boundary resources hinder or support development (Bianco et al., 2014; Foerderer et al., 2019). Bianco et al. (2014) identified several challenges the platform's complementors had related to the platform's boundary resources. They found challenges with the APIs, SDKs, and its documentation, such as limited use cases and difficulties getting started using the SDKs due to limited documentation. They also found that there were supporting factors for the complementors, such as that common API operations was quite straight forward, and interactions involving both the developers of the boundary resources and the platform complementors, such as hackathons and live chats (Bianco et al., 2014).

3.1.4 Knowledge boundaries

Another benefit of platforms and its ability to move some of the development to third-party complementors is that the complementors often have more insight into the situated context of use than what the platform vendor has. Von Hippel (2005) points out that bridging the context-specific information and the generic enterprise software information is challenging, but that it is necessary to develop software products that are desirable and feasible. Thus, complementors to the platform must know both the situated context of use as well as about the software architecture (Roland et al., 2017). However, expanding on the previous research on boundary resources, Foerderer et al. (2019) demonstrate that "platform strategies inherently impose 'knowledge boundaries' between platform owners and complementors" (p. 129). Furthermore, they argue that most contributions on boundary resources "relate to understanding how actions and reactions of platform owners and complementors shape resources at the boundary, rather than what influences gaps in knowledge across the boundary and how platform owners address these gap" (Foerderer et al., 2019, p. 121). In other words, it is found that platforms have knowledge boundaries, as complementors must know how to access and extend the platform functionality (Foerderer et al., 2019; von Hippel & Katz, 2002), which also includes the knowledge needed to be able to utilize the boundary resources provided. Acquiring platform-specific knowledge has been argued to be "one of the persistent problems for complementors" (Foerderer et al., 2019, p. 120). If the resources provided by the platform vendors are not sufficient, the third parties struggle with utilizing the platform. Foerderer et al. (2019) put forward their main findings, which is that the technical design of the platform poses knowledge boundaries between platform vendors and its complementors, to which platform vendors attempt to address the knowledge boundaries by providing additional boundary resources and through boundary-spanning activities. In relation to their findings, they argue that future research should investigate further the interplay between technological characteristics and knowledge boundaries (Foerderer et al., 2019).

3.2 Enterprise Software Platforms as Design infrastructures

To conceptualize all of the surrounding dynamics and resources that enable and support implementation of Enterprise software platforms, Li and Nielsen put forward a perspective of *Design Infrastructures* which "refers to both technical and social resources that may enable design after the initial design" (2019, p. 5). The design infrastructure consists of the collective practices as well as the material arrangements that make up the enterprise software platform and the surrounding socio-technical system. The surrounding socio-technical system consists of people, documentation, guidelines, channels of communication. The main purpose of the design infrastructure is to support design and innovations. It also covers all other activities, such as the design, development, maintenance, and implementation of the enterprise software (Li & Nielsen, 2019). In an enterprise software platform context, this also means that the design infrastructure also includes the platform's boundary resources as well. Hence, the design infrastructure covers broadly, from people and activities to technology and documents. Based on this definition, it can be argued that an enterprise software platform itself also *is* a part of the design infrastructure.

3.2.1 Generic-level Design and Implementation-level Design

Generic-level design and implementation-level design are processes that happen within the design infrastructure on different levels and with different aims. The processes also feed back into the design infrastructure. Both generic-level design and implementation-level design are involved in designing for use, by making artifacts or doing changes to artifacts that are intended to support end-users in their work (Li & Nielsen, 2019). The main difference being that implementation-level design is often concerned with a specific context of use with existing users and practices, whereas generic-level design develops artifacts to support a general user group or general use area. Additionally, the generic-level design also has processes that aim at developing artifacts that can be customized, to increase flexibility so that the enterprise software platform can become a better fit for specific contexts of use. This can also include more technical features, such as an application programming interface (API) that allow for external parties to create custom applications for the enterprise software. The generic-level can also provide documentation and take measures to build capacity on how to utilize these features (Li & Nielsen, 2019). In other words, generic-level design aim at developing structures and resources that can strengthen the implementation-level design processes.

4. Methodology

This chapter will first go through why an interpretive approach was most suited as a paradigm for the research. Next, the actors in the research context will be presented, followed by why case study was chosen as a methodology. Lastly, research methods used for data collection in the different stages of the research will be presented, as well as some ethical considerations and challenges.

Before the research started, I had already an interest in the new boundary resources being created for DHIS2, and I wanted to further investigate if and how these resources could be utilized in the implementation-level design of custom applications. This study is part of an ongoing, larger Action Research project, where

> "the primary goal of the HISP research is to *design, implement, and sustain HIS following a participatory approach to support local management of health care delivery and information flows in selected health facilities, districts and provinces, and its further spread within and across developing countries*." (Braa et al., 2004, p. 343).

This thesis aims at contributing to the ongoing project, by investigating the newly developed boundary resources; the DHIS2 Design System and the DHIS2 App platform. As mentioned previously, it is commonly known that it can be challenging to adapt enterprise software, and this is also a known challenge for the implementation-specialist groups when implementing DHIS2 for different use cases and in different local contexts. This thesis will investigate if boundary resources developed by generic-level design can fit in an implementation-level context, and thus be utilized in implementation level design.

4.1 The Paradigm and Methodology

The focus of this study has been a socio-technical system within a network of organizations and how they work with implementing DHIS2. The overarching aim was to understand the relationship between global resources and local practices. This will be investigated by looking at how, and if, the DHIS2's new resources related to app development can be utilized in the implementation of DHIS2. To do so, it is first important to get a deeper understanding of how the implementation process is currently done within implementation-specialist groups. It has been argued that organizations, such as the implementation-specialist groups, are created by humans "to serve their ends" (Susman & Evered, 1978, p. 584), they are made up of human action "in which the means and ends are guided by values" (Susman & Evered, 1978, p. 584). Organizations are complex social and dynamic entities, and they do not exist without humans and human interaction. The positivist paradigm "consider scientific knowledge to be obtainable only from sense data that can be directly experienced and verified between independent observers" (Susman & Evered, 1978, p. 583), in other words, the phenomenon studied must be possible to reconstruct and achieve the same outcome, and the result will be the same regardless of who it is observing it.

As I wanted to understand if and how DHIS2's newly developed app boundary resources could fit into the work practices of the organizations HISP India and HISP Tanzania, field trips were conducted to both locations to collect data for the research. As the research context is bound by time, and due to the ever-changing nature of organizations, it would not be possible to perform the same research twice to verify it as the positivist paradigm requires. Even given the same context and time, other researchers might have observed and interpreted things differently from what I did. Additionally, due to the exploratory nature of the research question, the research would not be meaningful to quantify. In essence, a positivist approach is not well suited, and it would not be able to explain the human actions within organizations, as it cannot capture the complexities, social dynamics, and the underlying social meanings. Hence, an interpretive, qualitative research methodology is best suited (Klein & Myers, 1999; Susman & Evered, 1978; Walsham, 2006). Interpretivism seeks to understand the underlying meanings, and it permits multiple correct approaches and findings. The interpretivist paradigm does not seek to find one objective truth but rather seeks to explain something about a socially constructed reality, in which the findings are mediated by time, context, and the researcher (Villiers, 2012).

Case study is a popular qualitative methodology within the interpretive paradigm. But there are also many other qualitative research methods, the most popular besides case study being ethnographic research, action research, critical research, and grounded theory. Some of the other types of qualitative research methods were not particularly well fitted for the research, due to the short time frame of the project. Ethnographic research requires the researcher to do longitudinal studies to gain an in-depth holistic understanding of the research subject and the context (Klein & Myers, 1999). Action research is conducted in cycles; diagnosing, action planning, action taking, evaluating, and specifying learnings (Susman & Evered, 1978). It is argued that to produce knowledge and practical contributions, the research should aim at completing multiple research cycles. Thus, following these methods could be difficult to achieve over a two-year period.

There are several approaches and definitions of case studies. Yazan (2015) discuss the work of three predominant case study methodologists; Yin, Merriam, and Stake. This research aligns best with Stake's work. Yin and Merriam's perspectives will be briefly presented, which will also explain why they do not fit this research. Yin's perspective on the role of the researcher is that the researcher should "maximize four conditions related to design quality: construct validity, internal validity, external validity, and reliability." (Yin, 2002, p. 3, referenced to in Yazan, 2015, p. 136). To ensure the validity and quality of the data, the researchers should keep these aspects in mind in every step of the investigatory process. Merriam suggests the case study should be thoroughly planned and follow a certain process in designing the research, which includes "conducting literature review, constructing a theoretical framework, identifying a research problem, crafting and sharpening research questions, and selecting the sample (purposive sampling)" (Yazan, 2015, p. 141). Due to the exploratory and interpretive nature of the research question, Yin's orientation towards positivism and Merriam's restrictions in flexibility, do not fit well with this research. In contrast, Stake argues for a more flexible design, that "allows the researcher to make major changes even after they proceed from design to research" (Stake, 1995, referenced to in Yazan, 2015, p. 140). He proposes four characteristics of case studies, which are that they are *holistic, empirical, interpretive,* and *emphatic* (Yazan, 2015). *Holistic* means that the phenomenon studied must be seen as part of the whole, in other words; it must be seen in relation to its contexts. It should be *empirical* as the researcher should base their study on the observations from the field. *Interpretive* in seeking to understand the underlying meanings, and "see the research as a researcher-subject interaction" (Yazan, 2015, p. 139). *Emphatic* means that the researcher should also attempt to reflect upon and understand things from the research subjects' point of view (Yazan, 2015). The four characteristics of case study which Stakes proposes fits well with the research conducted for this thesis.

4.2 The Research

Empirical data provided in this thesis was gathered over two years, from August 2018 to June 2020. I went on two field trips to Noida, India; the first field trip was in January 2019 and lasted for four weeks, and the second for six weeks and was conducted in October-November 2019. A brief field trip of one week was also conducted in Tanzania in May 2019. A benefit of the research being part of the ongoing HISP action research and the DHIS2 Design lab was that this provided me with access to the two implementation-specialist groups and the core team, due to the existing connections and collaboration between the HISP research group and the other contexts. Figure 3 shows a timeline of the events, both field trips as well as new releases from the core team. This context is important to note to better understand what "state" the Design System and the App platform was in when they were discussed during the field trips. It also illustrates how rapid new releases and changes were coming from the DHIS2 core development team, and how much had changed in-between the field trips.

Due to the long stays in India, I gained a more holistic understanding of the organization and all of its practices, whereas with the core development team and HISP Tanzania I primarily got to "zoom" in and explore the practices related to the research question. With the core development team and HISP Tanzania, I mainly had to rely on the data gained through what they said. I did not get the opportunity to participate and observe the practices in these contexts' over a longer period as I did with HISP India. However, much of what they said could often be strengthened by document analysis, for example by analyzing the software code and that it coincided with what had been said. Additionally, insight gained at HISP India also made it easier to understand HISP Tanzania and the core team, as a lot had already been learned about the DHIS2 software and the HISP organizations.

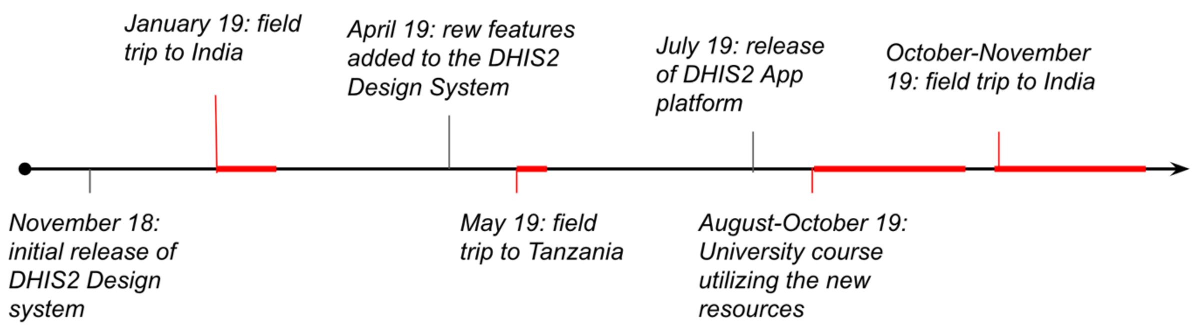

January 19: field trip to India
April 19: rew features added to the DHIS2 Design System
July 19: release of DHIS2 App platform
October-November 19: field trip to India
November 18: initial release of DHIS2 Design system
May 19: field trip to Tanzania
August-October 19: University course utilizing the new resources

Figure 3: Timeline of events in the project.

This section will first go through the methods used for gathering data from the various contexts; the DHIS2 core development team, HISP India, and HISP Tanzania. During all three field trips, two to India and one to Tanzania, I took notes from interviews, observations, as well as keeping a diary of what happened every day. Pictures were also used for documentation. Table 1 illustrates the order of the events, and Table 2 provides a summary of the data collection methods used in the various contexts, which will be elaborated on in the following sections.

Table 1: Order of events and data collection methods used.

Period	**Location**	**Activities**
January 2019	India	Interviews Workshops Participant observation Document analysis
April 2019	Norway	Interviews
May 2019	Tanzania	Group interviews and discussions.
August – September 2019	Norway	Document analysis Interviews
October – November 2019	India	Workshops Interviews

December 2019 – May 2020	Norway	Interviews (follow-up questions)

Table 2: Summary of data collection methods used in various contexts.

Core team	HISP India	HISP Tanzania
• Four interviews/meetings with Core developers, • Ongoing communication and follow-up questions, through their chatroom, • University course forum responses from the core team, and • Lectures held by the Core Developers in a university course.	• 30-40 very informal and unstructured interviews, • 17 more formal interviews (semi-structured and contextual), • 4 workshops focused on various topics, • Participant observation over two trips totaling up to 10 weeks, • Document analysis, and • Photos and Diary.	• Three full-day sessions of group discussions, • Document analysis of the libraries and apps made by the HISP Tanzania, and • Diary and photos.

4.2.1 Core team

I had contact with the core development team in Oslo throughout the whole project. Through the interactions with the core development team, I got a lot of insight into how and why they were building the resources they did. Feedback from HISP India and HIPS Tanzania was also shared and discussed with the core team after each of the field trips.

Data Collection Methods

Four formal interviews with core team developers were conducted. The interviews consisted of one to two developers and me, and in a couple of the interviews my supervisor and another master student were also participating. These interviews were semi-structured. A short list of questions was prepared before the interviews and had often derived from learnings in the field.

A channel in the core team's chatroom was created to plan together for a university course where I worked as a teaching assistant and to which the core team was a part of. In the university course, the students developed applications using the new resources the core team had developed. The chatroom was also used to easily ask the core developers about some follow-up questions I had during the research. A couple of the core team developers were involved in the preparations for the university course, bringing us up to date on their latest features. This was beneficial for us and them, as they got a large base of users to test their latest resources, and it was beneficial for us to get their support and knowledge. Two core team developers also held a lecture each, presenting DHIS2, the Design System, and the App platform. Some of the core team developers were also in the university course's online course forum. In this forum, the students would ask questions or post issues they had regarding the project, and either other students, teaching assistants, or the core team developers would answer the posts. During the student app project, this was extremely beneficial for all involved parties. The core team developers had a better technical insight into DHIS2 and the new resources than both teaching assistants and other students, and the students' questions and issues also gave the core team developers a better idea of what users of the new resources were struggling with, and where more documentation, tutorials, or examples was needed.

4.2.2 HIPS India

Through two long stays in India, a lot was learned about the HISP India organization and its practices. This was strengthened by the various data collection methods were used. I also gained a quite in-depth insight into one of HISP India's biggest projects. This included insight into the process, who was involved in the projects and various roles of people; ranging from the client to the HISP India team and down to end-user, details on how communication flowed; who communicated and what was communicated between the different roles. A lot of insight was gained into the practices surrounding the development of applications as this was a main area of focus. Additionally, I also learned a lot about other practices and much about other activities HISP India do in relation to DHIS2, such as configuration, end-user training, integration of DHIS2 with other systems, custom development of reports, and challenges related to various DHIS2 versions and upgradability. Through this, and due to the long time spent in India, a lot of knowledge was gained on the broader organizational culture of HISP India.

Data Collection Methods

Due to the long field trips and many data collection methods used in India, the different methods will be presented one by one in the subsections.

Interviews and Workshops

Most of the interviews on the first trip were very informal, being unstructured, and more reminiscent of ordinary conversations (Zahle, 2019). This was beneficial because the developers were working under quite immense time pressure, and it was easier for them to stop and take a five-minute chat with me, rather than having to plan it and set of time for me. At the same time, I could guide the interviews towards the topics of interest. Some discussions were done during lunch, others over a brief coffee break. As these interviews were very informal, they were also hard to count, thus they are not included in the overview, but it is estimated that these account to about 30-40 interviews. These very informal interviews were not recorded, but notes were taken during the interviews. The more formal interviews are summarized in Table 3.

As HISP India has many clients, we were invited to take part in one of their bigger clients, Alpha Consulting, and use this as our case for research. During the first trip, me, my supervisor, and a couple of the other master students got to interview representatives from Alpha Consulting. We also did field visits to hospital facilities, where we interviewed different end-users. Contextual interview was then an appropriate method, as we could ask the data entry personnel to show us how they would perform their tasks. While they demonstrated their practices for us, we would ask follow-up questions when necessary. On the trips, we had with us a representative from Alpha Consulting as well as one of the HISP India employees.

Figure 4: Contextual interview of end-user at a hospital facility in Uttar Pradesh.

On the second field trip to India we no longer had access to Alpha Consulting due to some bureaucracy. Thus, I did a lot more formal, yet still semi-structured, interviewed with the HISP India team. These interviews were conducted by me and one of my fellow master students. We had prepared an interview guide, with some topics and questions we wanted to ask. This was to enable discussions, and follow-up questions from both us, the interviewers, and the participants. The interviews were mostly conducted in a small meeting room in the office, closed off from the open landscape where the HISP India team worked. Before the interview started, we would inform the interviewee about the purpose of the interview as well as how the data would be stored and used, before we asked for their consent. These interviews were recorded, so we asked if this was ok for the interviewee. We had quite similar interview guides for interviewing the implementers (the public health group) and developers (the technical group). However, the follow-up questions would be quite different, as the interviewees explained the process from the perspective of their role and their responsibilities. Almost all the interviews started with asking the interviewee to tell us about a project they had worked or were working on. We would follow this up with questions about who was involved in the projects and about the general process in the project from start to end. We asked the interviewee to describe their role in these projects, and whether or not there was anything they would want to change about the process. We also asked them about the challenges they had faced in projects, who else was involved from the HISP India team, and how they would communicate within the HISP India team as well as with the client. Lastly, we would end by asking how long the person had worked for HISP India.

Additionally, a total of four workshops were held. The themes for the workshops were: two workshops focused on design techniques, one held by a core team member on the DHIS2 App platform, and the last about local design and usability. The length of the workshops varied, the first two were each held in one day, the App platform workshop over two days, and the last spanned over a full workweek. In the workshops both implementers and developers participated, except for the App platform workshop where only the developers participated.

Figure 5: Workshop with HISP India, in Noida, India.

Table 3: Interviews and workshops (very informal interviews not included) conducted during the two visits to India.

Activity	Total activities conducted	Participants
Workshops	4	• Developers HISP India • Implementers HISP India • Master students from UiO • Ph.D. candidate from UiO • Developer from Core team
Interviews	**First trip**: • HISP India office: 4	• Developers HISP India • Implementers HISP India

	• At Hospital facilities: 3 **Second trip**: • HISP India office: 10	• Data Entry Operator (DEO) • Monitoring and Evaluation (M&E) Officer • Researcher at HISP India

Participant Observation

Participant observations consisted of both observing how the developers were working as well as participating in meetings, workshops, and other events that unfolded in the office. Most of the participation was done in what Zahle (2019) describes as a *weaker sense*, we would hang around in the office and try to observe and take note of what was going on. This was done to get a better understanding of how the HISP India team worked and a better understanding of the work environment and the context.

Document Analysis

Document analysis was used extensively throughout the research. Before the introduction of the new resources provided by the core team, a thorough document analysis was conducted of all the custom applications made by the HISP India team for Alpha Consulting. This was done to get a better understanding of how the team developed applications. Throughout the research, I also did document analysis of the source code the developers had made, as this allowed for follow-up questions about for example libraries used by the developers, and a better understanding of the development practices.

Photos and Diary

During both trips to India, I kept a diary. Each day was described, from what had happened and been done in the office that day, to the events that unfolded in our everyday life. A lot of photos were also taken during the trips, documenting many aspects of the trips. The diary and photos were very helpful, as they made it a lot easier to recollect things after the visit, such as when certain events unfolded. It also helped in remembering the context.

Context and Challenges

Due to the long time spent in India, my fellow master students and I gained a deep understanding of the HISP India organization and their work practices. We also got a good understanding of the culture in the HISP India office. On our first trip, which lasted for four weeks, we became quite well acquainted with the HISP India team. In-between the two trips, we also stayed in contact with the team. This made it easier when we came back for the second trip lasting six weeks because we could then "slide" back into the team and could go more directly into the research, without all the formalities when being new to the context. At the same time, it was beneficial to have a break between the two field trips, as some distance can make it easier to analyze and better understand what we have learned.

Even though we felt we had somewhat understood the cultural context in the HISP India organization, there are still many things in the broader Indian culture we did not understand, which can have been barriers to our understanding. The language was of course a challenge. Even though almost everyone in the organization spoke English very well, a lot of interesting information may have been lost, as they would often switch to Hindi in discussions during for example the workshops, and we did not understand what they were talking about. The language barrier was most evident during our sessions with end-users at the hospital facilities. As most end-users did not speak English, either the Alpha Consulting representative or HISP India employee would translate for us. Sometimes the user would talk for a long time and have discussions with either the HISP India employee or Alpha Consulting representative, and in the end, we would get a short summary of what had been said.

As mentioned, we did get quite well acquainted with the HISP India team. However, they had a quite high turnover of developers. During the first trip, there were about 7-8 developers in the team. On the second visit, only three of the original developers remained (one of which was located in Lucknow, India), and they had four new hires that had worked there for a couple of months. This led to us becoming best acquainted with implementers, but also the developers who had also been there on the first trip. Besides spending time together at the office, we also did some social activities together with the HISP India employees, such as visits to markets and eating out at restaurants. However, in the office the HISP India employees would often refer to us as “ma’am” and “sir”, a term often used to express respect, often used when referring to superiors. This was also noticeable within the organization in general, as the team would also use these titles when talking to or talking about someone superior to them in the office.

Even though we sometimes felt that we had “become one of them”, it was still clear that there were still some cultural barriers. One example was when celebrating the holiday Diwali in the office. The office was divided into teams, and each team was going to make a *Rangoli* (creating patterns on the floor with colorful powder). It was hard to follow what the plan for the Rangoli was, as a lot was said in Hindi and no one shared any plan. One of my team members disappeared just after we had been assigned into groups, and it was not until the end I realized that he had been in another room the whole time painting a jar that would be placed next to the Rangoli. So even though we mostly felt that we had a good understanding of what was going on, there were still these small events reminding us that there was still a lot that we did not understand.

Another challenge we faced, was that we knew little about the domain when we visited on the first trip. During conversations, interviews, and discussions, many domain-specific terms we did not know the meaning of were used. The HISP India team and Alpha consulting would use these terms often assuming we understood, or not considering that it could be unfamiliar to us. We would write down what we heard, and later on, we structured our notes on a whiteboard and discussed it with some of the HISP India employees. This would often clarify what parts we had not understood correctly, and the HISP India employees would explain. An example of this was from our field trip to some Hospital Clinics where the interviewees kept talking about “emeny”. It took us a quite long time to realize that they were discussing “M & E”, which stood for Monitoring and Evaluation.

Lastly, we also experienced that environmental factors in India were challenging. During our second trip to India, the air pollution in Delhi and Noida was record high. The extremely high levels of air pollution, which was rated as “hazardous”, put a great toll on us. We had headaches, felt drowsy, and lacked energy in general. During this period, many of the HISP India developers and implementers would take days working from home. This was also challenging, as they were unavailable for follow-up questions, and sometimes we would have to wait for days before they came back to the office. The air pollution affected both our work as well as that of the HISP India team.

Figure 6: The record-high air pollution in India had an impact on us, our research as well as the HISP India team.

4.2.3 HISP Tanzania

A brief field trip to Dar es Salaam, Tanzania, was conducted in May 2019 by me and my supervisor. We spent three days at their office, with a day in-between the first and the second visit. The first day was spent on going through the applications they were currently working on, and their implementation involvement and process. They also presented the libraries they had developed to support their app development. On the second day, we held a Design Workshop where my supervisor introduced some research concepts, the themes were: usability, user experience (UX), users and user participation, methods and techniques to understand and involve end-users, and analysis and prototyping. In the end, we discussed how these themes related to HISP Tanzania's work process. On the last day, we discussed the commonly used approaches to tailoring DHIS2. We discussed current work processes, and which tools they used in the development of custom applications.

Data collection methods

The sessions mainly consisted of group discussions. The team at HISP Tanzania, as with HISP India, consisted of both developers and implementers. The group discussions consisted of us, and the five developers at HISP Tanzania. Each of the sessions lasted about a full day. In-between the sessions, I conducted a document analysis of the libraries and applications published by the HISP Tanzania team, which was also presented and discussed with the HISP Tanzania team. During this trip, I also kept a diary and took some photos of the unfolding events.

Challenges

It should also be mentioned, that due to the short time spent in Tanzania, the same insight was not gained in Tanzania as within HISP India. I only got a "sneak peek" into the HISP Tanzania organization. I did not get the opportunity to observe their work or come with follow-up questions in the same way as with HISP India. Therefore, there is a lot that can have been omitted regarding the findings from HISP Tanzania. However, the learnings from HISP India provided a basis to better understand much of the HISP Tanzania team's context, as a lot had already been learned about the health domain, the DHIS2 and its tools and resources, as well as common practices and challenges.

Figure 7: Group discussions with the developers at HISP Tanzania.

4.3 Data Analysis

Due to the interpretive, qualitative nature of the research, the data analysis is an important aspect. Anthropologist Clifford Geertz argued that "what we call our data are really our own constructions of other people's constructions of what they and their compatriots are up to" (1973, p. 9). It is thus important to emphasize that the findings are based on my interpretations and understandings. However, during the data collection, patterns started to emerge, and the findings from the various data collection methods supported each other, which further strengthened the quality of the data. Many of the findings were discussed with the participants. This was beneficial as they would often supplement the findings or could rectify misunderstandings. This further strengthened the validity of the findings.

A hermeneutic approach to understanding was natural due to the exploratory nature of the research. Thanks to the practical learnings shared from the Design Lab and theoretical contributions in the field, I started with some pre-understanding of the field. Prior to the field trips, I already had an interest in resources for developers and was interested in the participants' practices in developing applications. Once getting a bit familiarized with the HISP India organization as a whole and its practices, I started attempting to understand the various practices related to developing applications. However, it quickly became evident that I needed to understand more about the broader context and other practices related to the implementation of DHIS2. Once a better understanding of the whole was gained, the focus could return to the specific parts of interest. The research went on like this throughout the project; going out to investigate the broader situation, gaining more understanding, and then zooming back into the various parts, getting a better understanding of the whole, and repeating the cycle. Even though the practices of application development were the main interest, it was not possible to understand or ask the right questions about it without a broader understanding.

The learnings also went in cycles in relation to the various contexts from which data was collected. Combined, the learnings from HISP India, HISP Tanzania, and the core team provided not only insight into their practices, but also a broader understanding of the various actors and activities that surround the DHIS2 as a whole. Each learning from the parts provided a better understanding of the whole, which again would make it easier to understand other parts or get a deeper insight into the parts. Figure 8 illustrates the analytical process followed in the research.

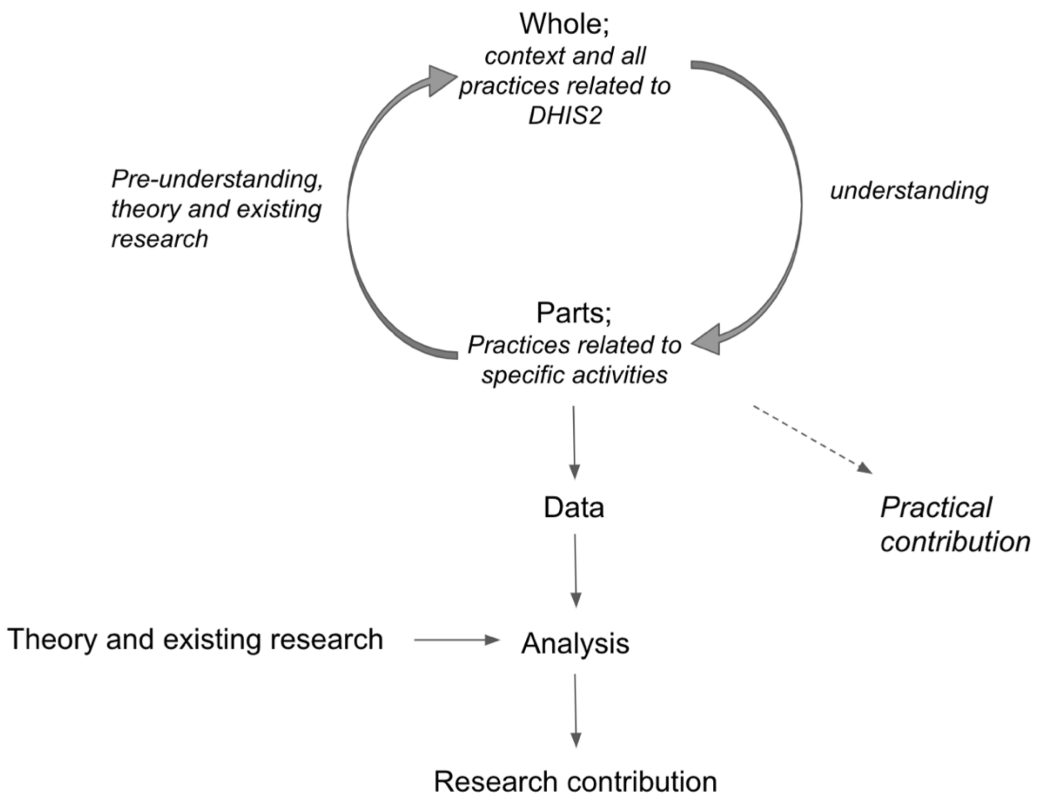

Whole;
context and all practices related to DHIS2
understanding
Parts;
Practices related to specific activities
Pre-understanding, theory and existing research
Data
Practical contribution
Theory and existing research
Analysis
Research contribution

Figure 8: Analytical process for the research, based on hermeneutics.

5. Findings

This chapter will present the findings of my research through five main sections focusing on;

1) DHIS2 resources to support implementation-level design,
2) Practices of current app development during implementation-level design,
3) Local resources related to app development,
4) Challenges with app development during implementation level design, and
5) the core developers' new boundary resources aimed at strengthening the design infrastructure around application development.

The findings will be presented based on the topics, as it helps highlight the different activities surrounding application development, both during implementation-level design and generic-level design. As the findings are presented in this way, a timeline is provided in Figure 3 in Chapter 4, to show the chronological order of how the events unfolded.

The findings presented will provide a basis for the analysis in the next chapter. The findings will start with a description of the existing practices of application development during implementation-level design within the two organizations HISP India and HISP Tanzania. Following this, I will explore the local resources that have been developed by the two implementation-specialist groups to strengthen the implementation-levels capacity to innovate. I will then present the current challenges the two implementation-specialist groups are facing related to developing custom applications. The next section will focus on the platform resources the core developers recently have introduced to the design infrastructure to make it easier to develop applications for the DHIS2 platform, including an user interface component library and an app development resource platform.

5.1 DHIS2 resources to support Implementation-level design

This section will present some of the boundary resources provided by the DHIS2 core development team for the DHIS2 design infrastructure. The DHIS2 core team has developed a range of resources to better support implementations of DHIS2. Among these resources are documentation of different aspects of DHIS2, there are several different developer guides, configuration guides, tutorials, demonstrations of the software, community forums, security procedures, and information about cloud hosting. The core team also takes part in DHIS2 academies, where they train implementers and developers on different parts of DHIS2 implementation. The academies' themes range from Data Quality or Analytics tools to more technical aspects such as Server Administration or Web App Development.

Additionally, the core team develops and maintains the DHIS2s core code and the bundled applications. The bundled applications are designed with configuration options and aimed at being used without much effort besides configuring, for example configuring the data elements needed for a data entry application. The core team now has developed and is maintaining about 34 applications and 6 libraries. These are supported in four versions of DHIS2, which means that they have a lot of code to maintain.

Two of the libraries previously developed by the core team aimed at strengthening the development of applications is *d2* and *d2-ui*. D2 is a JavaScript library to abstract away complexities of doing DHIS2 API calls and provides models to communicate with the DHIS2 server. The d2-ui library offered UI components, ranging from simple components such as buttons, progress bars, or select fields, to more complex components such as a header bar and organization unit trees. The header bar has to be integrated with the DHIS2 instance, for example getting the DHIS2 instance name and showing the user available apps in the specific DHIS2 instance, whereas a button has no dependencies. In other words, the d2-ui consisted of both integrated and freestanding components. In an effort to improve the resources the core development team developed new resources based on the existing d2 and d2-ui.

Efforts to further strengthen both internal and external app developments were made by developing two new resources; the DHIS2 Design System and the DHIS2 App platform, which will be elaborated on in section 6.5. Due to the initiative of these resources, the d2-ui was discontinued and replaced by the DHIS2 Design System. The d2 library persisted, but the new App platform made it obsolete.

5.2 Practices of App Development during Implementation-level design

The following section will present the current practices and processes in HISP India and HISP Tanzania, particularly related to how they currently develop applications. In other words, it follows the organization’s implementation-level design process. The implementation processes will be described broadly, from the start of a project to their approaches to find a solution, as well as *when* it is the implementation-specialist groups decide to make applications and the practices of how they make the applications. This information is important to understand and analyze how the new resources can fit into the implementation-level design process of the two implementation-specialist groups.

5.2.1 HISP India

When and Why Applications Are Made

HISP India has many ongoing projects, the projects are mostly centered in Asia, yet they do have projects across the globe. Their clients range from government ministries to NGO’s. HISP India gets new clients through usually two approaches. Sometimes clients approach them and ask if they can take on the project. Other times they find projects that they feel can fit well with DHIS2, and they bid on the contract. Once they have gotten a new client, an implementer assigned to the project starts gathering the requirements. The requirements are mostly focused on what kind of output values the client wants. Once they have the requirements, the implementer starts mapping the outputs, figuring what kind of data input values they need to generate the clients desired outputs. After that, they configure DHIS2 for the required inputs. If the requirements cannot be addressed through the standard configurability of DHIS2, the implementer asks for support from the lead developer. HISP India often attempts to negotiate requirements with the client if their requirements are not possible to configure. If this happened, the HISP India team would sometimes attempt to make a Jira ticket to the core team, expressing requirements, configuration options, or features they wanted. However, they rarely received any response from the core team. One developer explained that he had put in a request 6 months ago, but no change or response had come of it yet. So, if the client is persistent in their requirements, and it is not possible to deliver a solution through configuration, the team develops a custom application for them. One of the developers explained that they were needed "*when there is a roadblock. The configuration team they give up. They say 'no, this is not possible anymore, now we need some help'* ", and it was at this point a developer was brought onto the given project. Configuration was the preferable option, as custom applications took a lot more time and hence, were more costly. The HISP India team stated that the client does not care if their solution is a custom application or achieved through configuration as long as it fulfills the requirements. The HISP India team would go to great lengths to make a client's solution through configuration, but if a configuration was not possible, they would develop custom applications for the client.

Usually, one developer is assigned to the task and works alone on developing the application, however, the developer works together with an implementer. The implementer communicates and gathers the requirements from the client, and forwards these to the developer. Once the application is made, the solution is handed over to the client. If the solution is approved, the developer gets reassigned to other tasks or projects. The implementor is often the one to train someone, either the actual end-users or more often some managers or representatives from the client organization, who then is responsible to further train the end-users. If the application is not approved, the developer changes the parts the client wants to change. It is not uncommon that the developer sometime later gets new requirements from the client and must do new changes in their applications. Hence, most work HISP India do regarding applications, are changes in existing applications.

Another reason for why the HISP India developers had to develop custom applications was when bundled applications changed. The core team would regularly update the bundled applications with new features and user interfaces. If features used by clients of HISP India were removed from the applications, the HISP India developers would sometimes be forced to customize the application to make these similar to the old version of the bundled application. This had happened with several applications used in the UPHMIS project, and Alpha Consulting had requested several custom applications be made.

How Applications Are Made

As mentioned, developers at HISP India usually work alone as the only developer on a task or project. For this reason, the developer can quite freely choose the technologies and frameworks to be used in developing the application. When starting on a new application, the developer assigned to the task receives the requirements the implementer had gathered from the client, which mainly consists of a list covering the functionality which the client wants. There are rarely specific requirements for the UI. One of the developers said that he often sketched the UI of his application on one of the whiteboards in the office before he started working on it, but that he was the only one doing this type of prototyping. During a workshop, one of the developers said that "*we are mainly backend developers, no experts on frontend and making good interfaces*", and explained that he had looked for university courses for learning UX design, but that he had not found any offered at any university in Delhi.

The developer usually starts with a *fork* of an existing app. When doing a fork, the application's code and everything else that might be in its repository are copied into another repository. From this new repository, a developer can do changes in the code without affecting the original application. The HISP India developers would often fork bundled applications the core team had developed, and they had one bundled app, in particular, they forked very often. They had also made forks of the bundled applications which they did not modify. They did this to keep older versions of the bundled applications in their repository, as their fork of it would not be updated even if the core team updated their bundled applications. The application the developer starts with is therefor often found in HISP India's GitHub repository. In the repository, they have sub-repositories consisting of all the applications they have made, as well as forks of bundled applications. As one developer said "*we take a structure and we modify it. Usually a structure we have already made*". Once they have found the most fitting existing application, the application is forked from their repository. For the bundled applications, this meant that they would fork an application that they had previously forked into their repository from the core developers, and then do changes to the code to make it fulfill the requirements of the client. Other times, if the forks of the bundled applications or their own previously developed custom applications are too different from what is wanted, they look for other options, such as applications developed by other implementation-specialist groups of DHIS2. The process is illustrated in Figure 9. This shows how the HISP India team utilize existing applications when developing solutions for their client. It also shows how the custom applications developed goes back into their repository and can be utilized in future projects.

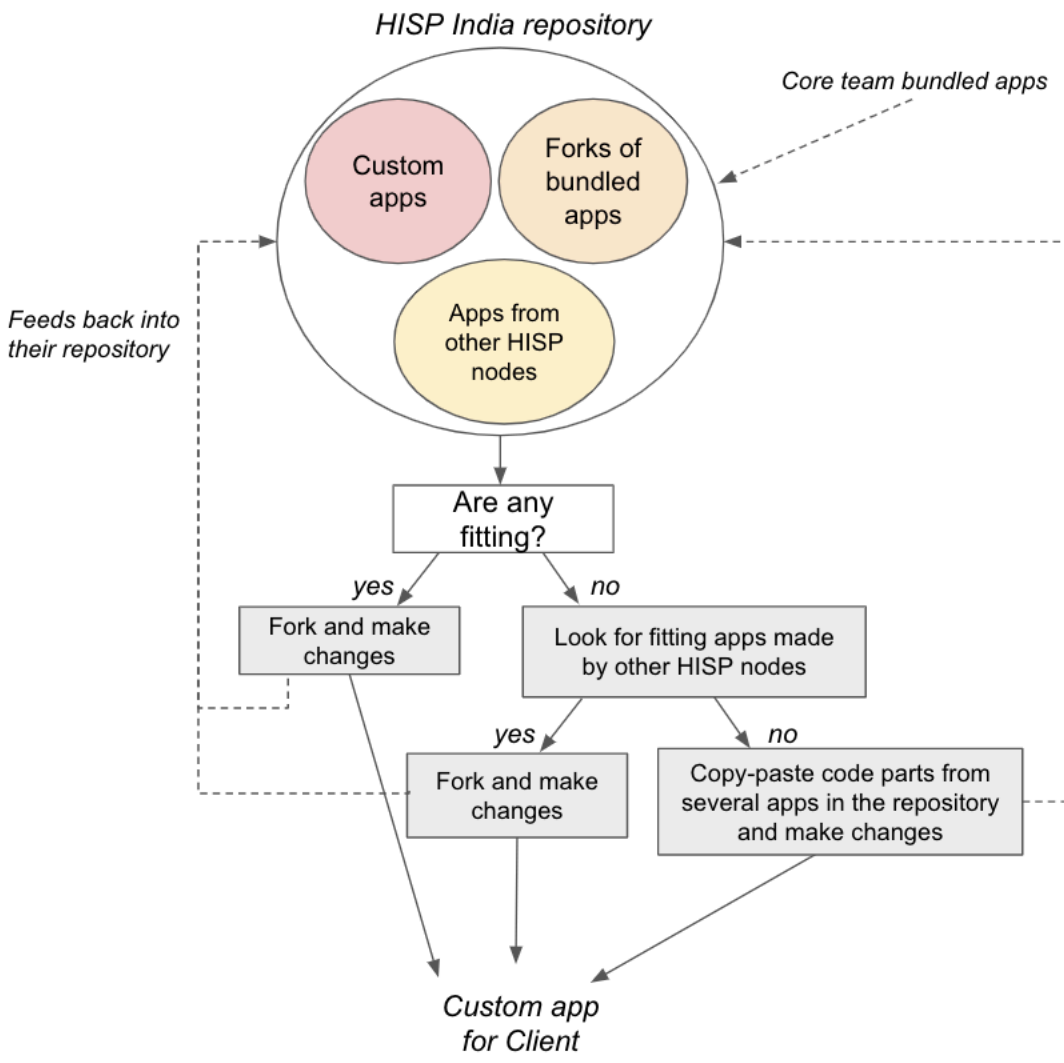

HISP India repository
Core team bundled apps
Custom apps
Forks of bundled apps
Apps from other HISP nodes
Feeds back into their repository
Are any fitting?
yes
no
Fork and make changes
Look for fitting apps made by other HISP nodes
yes
no
Fork and make changes
Copy-paste code parts from several apps in the repository and make changes
Custom app for Client

Figure 9: Process of how developers at HISP India create custom applications.

5.2.2 HISP Tanzania

When and Why Applications Are Made

As with HISP India, HISP Tanzania also preferred configuration over the development of custom applications, as the development of custom applications was very time consuming, and hence also had a higher cost. They explained that configuration was usually enough when implementing an HMIS, but when they were working with other domains, they would almost always have to create custom solutions. The team had ongoing projects in many countries and within an increased variety of domains, such as agriculture, water programs, sanitation, social welfare, and several others. When configuration was not possible, they would see if any existing applications could match, otherwise, they would see if any bundled applications could be customized to fit.

If it was decided that a custom application was needed, they had mainly two different ways of starting. If they had experience in making solutions for the given domain they were making the application for, they would start the development based on the client's requirements. However, if the domain was new to them, the developers, together with implementers, would start by talking with a domain expert of the field for which they were making the application. In the first meeting, they often did not have any particular questions for the domain expert, but they mostly asked about things that came to mind during the discussions. The aim of these meetings was to understand what kind of applications or solutions the HISP Tanzania developers could make for the client and to understand what was needed to meet what the client was thinking. After getting an understanding of what was needed, they would often start sketching a solution on paper. The developers would then discuss the sketches with each other. Internal discussions and sketching of prototypes would be done in several iterations. Once they had reached a sketch they were satisfied with, they would start developing the application. They described their meetings with the domain experts as progressive. As they developed, questions would emerge. They would then bring their prototype to the client and ask them about things and get feedback on the solution. Ideally, they would also like to show and discuss the prototype with the end-user. However, this was usually not possible due to limited time and funding in the project.

Once the application was accepted by the client, the developers would often be involved in training the end-users. One developer stated "*Actually in training, if any person has developed a particular feature, they have to be there in the training. [To] see how the solution works*". They explained that their role was not only being developers but also implementers. During these training sessions, the developers would observe the users and what they struggle with in the applications. If they experience that many end-users struggled with the same things, they would sometimes do changes in the applications later on. The experiences gained through training end-users were also beneficial for them when they developed applications. Even though they most often did not have access to end-users in the development phase, they would use their experiences to try to put themselves in the end-users shoes and keep this in mind when developing an application. One developer expressed that they had "*acquired advanced skills through training [end-users]*", both in understanding the end-users, as well as understanding many domains for which they had developed solutions for. They argued that the experiences gained from training end-users could somewhat compensate for the lacking involvement of the end-users in the development phase.

HISP Tanzania would not only develop applications for their clients, but they would sometimes also develop applications "for fun" as a contribution to the DHIS2 community. They would take some time to be creative and make innovations that could be used in the future. These applications have been developed generic enough to be used by many, and often have configuration options to adapt it to different contexts. These applications, as well as other custom configurable applications developed for clients, have been published to the DHIS2's online application "store", as illustrated in Figure 10. Publishing the applications to the online store makes them available for anyone to download to any DHIS2 instance.

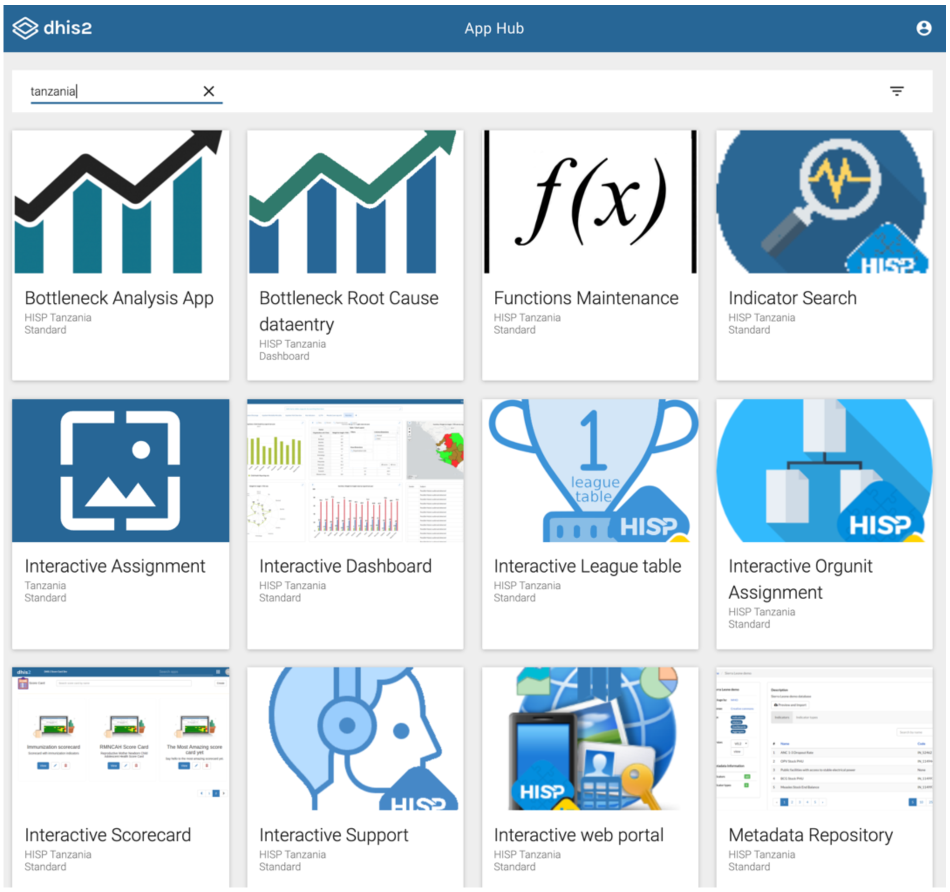


Figure 10: Applications created by the HISP Tanzania developers published on the DHIS2 online "store".

How Applications Are Made

As with HISP India, HISP Tanzania had often forked bundled applications and made changes to it to get more flexibility than what the bundled application offered to fulfill the client's requirements. When deciding between forking a bundled application or creating an application from "scratch", they considered what was most feasible, and also the time it would take. Usually, if the solution the client is looking for was similar to some of the bundled applications, they would fork it and add functionality to serve the specific case. However, as one developer stated, "*if we have to bridge together more concepts from DHIS2, then we start from scratch*". Timewise, they estimated that there was not much difference, as it took time to familiarize themselves with the existing code in the forked application. They also considered the long-term effects, such as if the application could be used in the long run and future maintenance work. They explained that the changes were not hard to do, however, the most challenging was maintaining the application. They also took into account whether or not they thought the application to be developed could be used in other places later on. If it could be useful later on, it was better to create a custom application, so that it could be used to solve other similar problems. They preferred to create their applications from scratch, as this allowed them to choose the technology and frameworks supporting the application.

When developing applications, one developer expressed that they would "*try to think ahead and create an environment that can support changes. And also take into account future maintenance and changes*". For example, some applications had been custom developed for a client with specific requirements. This application had been developed, yet with many configuration options, making it fulfill the requirements for the specific client as well as making it usable for others in the future. This also made the application easier to adapt to changed requirements from the client. This design coupled with a document explaining step by step how to configure the application, made it possible for the others to configure the application themselves without the support of the HISP Tanzania team. In addition, the developers created a demonstration to further help the user to configure the application as he or she wanted without needing any programming skills.

5.3 Local Resources

As illustrated in the last section, there are some differences between the processes of which HISP India and HISP Tanzania develop applications even though they develop applications for the same software and have many similar projects. This can be due to many factors, such as different types of clients, difference in funding, and capacity. However, to make the development of applications easier for themselves, the two DHIS2 implementation-specialist groups have developed a range of resources to support the development of applications. This section will focus on these resources they have developed themselves. Both implementation-specialist groups had developed their own local resources and practices, yet, they had done so in a quite different manner. This need to develop their own resources could suggest that the DHIS2 design infrastructure provided by the core team is inadequate and that the local resources and practices might be an initiative to compensate for these shortcomings.

5.3.1 Local Resources at HISP India

At HISP India, the developer working on a given application can freely choose to use any framework and libraries they deem fit for the project. Going through the source code of many of the custom applications developed by the HISP India team shows that there is a variety of external libraries and UI components used in the applications. One developer expressed that she would "*never write a whole code for this [showing me an application]*", and that as long as there was a library that could do what they wanted to achieve, they would always choose to use the library rather than writing something from scratch. Many libraries are available online and provide simple, freestanding UI components, such as buttons or checkboxes, so libraries were often used for these purposes. However, DHIS2 specific components, such as the DHIS2 header bar or the organization unit tree, shown in Figure 11, were often mentioned to be challenging. For these components, the developers would often copy-paste code of the header bar or organization unit tree from another one of their projects. Especially the developers who had worked there for a longer time, knew a lot of the code basis in the existing applications and knew which application to go to in order to find the code parts they needed. One of the most senior developers had also developed a couple of his own components, among them the organization unit tree, which he published to a package manager software, and frequently reused. As they could freely choose which libraries they used, it differed a lot among the developers. Each developer had over time developed their own set of resources that he or she would use when developing applications, such as preferred libraries or existing code they liked to take code parts from. Many developers expressed that once they had come to familiarize themselves with some code or library, they would prefer to continue using it whenever possible, as getting familiar with other's code or components took a lot of time.

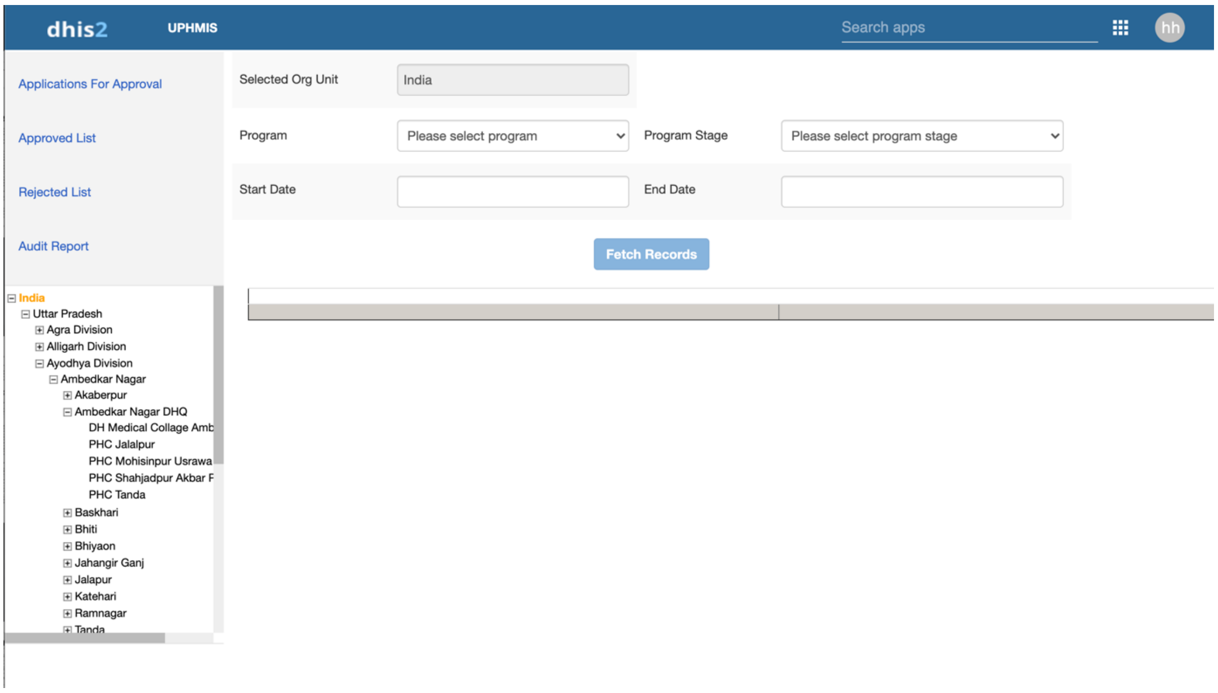


Figure 11: One of the applications developed by HISP India.

HISP India's GitHub repository, mentioned in the last section, can also be seen as a part of the design infrastructure. They keep all their existing applications in the repository and frequently reuse existing code structure as a starting point when developing new applications. They also reuse parts of various applications in the repository, as the code parts they need are copied from existing applications and pasted into new applications.

5.3.1 Local Resources at HISP Tanzania

The HISP Tanzania developers have developed many custom applications for clients but also more generic applications not developed for any particular client, but rather for the DHIS2 community. The team described that they had a recent initiative which they called *"componentizing"*, where they aimed at creating reusable components. They created several libraries of their own, consisting of useful components to support the development of applications. All their libraries were published on a package manager website, which makes it freely available for anyone to use. They had two processes of how components were made, which are illustrated in Figure 12. One of the developers expressed that innovations came in two ways, "*either a problem looking for a solution, or a solution looking for a problem*". When developing applications, they would try to think ahead on what could be reused and develop the application as modular as possible. If they later developed an application that needed some of the same functionality, they would extract that part of code and create a component of it. They would then publish the component to one of their package libraries to the package manager software and change the two applications so that they imported this, now external, component. The other process in which components were created, was by the developers creating a component they anticipated would be useful in the future. This was, of course, no guarantee for the components being needed. However, these components often ended up being used. For each component developed, they would create a README file that explained how to configure and use the given component. When they worked on the components, they would consider the implications of every new feature. If too much changed or the component became too complex or specific, it could affect the reusability of the components. The developers focused a lot on making the components with many configuration options. This made the components flexible, and it made it easier to avoid too many components. The goal with the component libraries was to reuse them, as it shortened the time and reduced challenges they had when implementing an application for DHIS2.

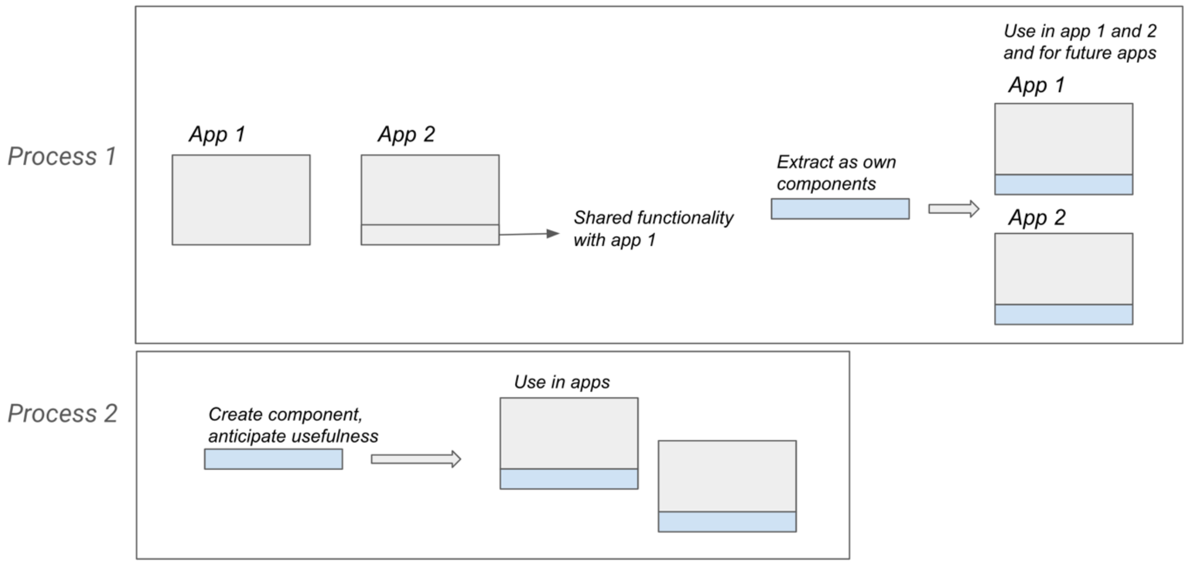
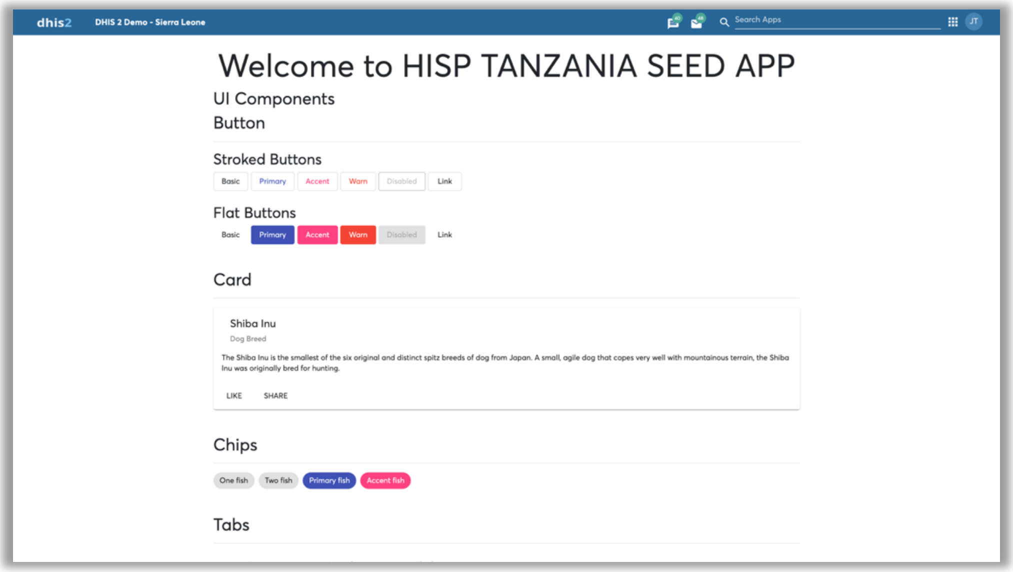

Process 1
App 1
App 2
Shared functionality with app 1
Extract as own components
Use in app 1 and 2 and for future apps
App 1
App 2
Process 2
Create component, anticipate usefulness
Use in apps

Figure 12: Illustration of the two processes in which new components were created by the HISP Tanzania developers.

The components developed by the HISP Tanzania team consist mainly of *functional* components, rather than UI components. The reason for this was that the "*customers are not so concerned with the UI*" as one developer stated. Thus, for the UI, they had in almost all of their applications used the popular, free-to-use, UI library *Angular material* made by Google. They had also developed components that combined functionality and UI. For example, their organization unit tree-component both had an UI element showing the tree structure, as well as the API services needed to get the organizations from the DHIS2 instance. Another example was when they needed a menu for the UI. They asked the core about a menu component they had previously used, but it was no longer maintained by the core team, so they ended up making their own menu component. Almost all of their components were developed to be used in the Angular framework.

In addition to the development of libraries and components, the developers had also developed a *Seed app*, illustrated in Figure 13. The Seed app is a sort of template application created with Angular framework, consisting of UI elements from Angular material. The application serves as their starting point when developing new applications. A lot of the setup needed to connect an application to a DHIS2 instance was already in place, with instructions on how to set it to your chosen instance. The application consists of all the parts they considered shared by all DHIS2 applications such as the header bar, API services, loading screens, and so on. The application was also documented with a README file, explaining how to install and start running the application.



The Shiba Inu is the smallest of the six original and distinct spitz breeds of dog from Japan. A small, agile dog that copes very well with mountainous terrain, the Shiba Inu was originally bred for hunting.

Figure 13: HISP Tanzania's Seed app

As with HISP India, HISP Tanzania also reused existing code in their GitHub repositories. However, they had organized it a bit differently. They had two main repositories; *iapps* and *hispTz*. Within the hispTz repository, they organized the source code of projects they had implemented. However, in iapps, they had all their generic innovations; both generic applications as well as components. This made it easier for them to find the reusable parts. This also made the components easier to maintain, as the components would only have to be updated in the iapps repository. When updating in the repository, changes would be updated in all the applications using the components. This was because they did not copy and paste the component into other applications, but because they published them to the package manager and imported the components into their applications.

HISP Tanzania mainly worked on implementing DHIS2 and creating solutions for clients. Despite this, the developers had not only developed their local resources for themselves to use but also for others. Some of the resources had been developed for other developers. One of the HISP Tanzania developers explained that one of their libraries had been developed for the HISP Uganda developers, he stated that "*We as developers are used to writing [code] for developers in other countries. Developers as users*". Hence, the HISP Tanzania group is also doing activities related to generic-level design, where they were providing resources aimed at supporting others in developing applications in implementation-level design. The resources and libraries they developed were mainly aimed at being generic, open, documented, and available for anyone to use. These resources often served a dual purpose, supporting both their app development, as well as those of other app developers for DHIS2.

5.3.2 Summary of Local Resources

To summarize, the resources utilized by the two implementation-specialist groups consisted of many resources besides those provided by the core team. Table 4 highlights the different local resources utilized by the two implementation-specialist groups in the development of custom applications. It should also be mentioned that the two implementation-specialist groups have developed applications in different quantities, and with differing purposes. Both of the groups have developed applications as custom solutions for specific implementations. However, HISP Tanzania has developed a larger amount of applications compared to HISP

India. Many of HISP Tanzania's applications have also been developed to be generic and aimed at being used by many, rather than being developed as a custom solution for a specific implementation.

Table 4: Resources utilized by the two implementation-specialist groups in custom app development.

HISP India	**HISP Tanzania**
• Libraries and structures provided by the core team, e.g. d2-UI and bundled apps • External UI libraries • Some self-made, reusable components • Code structures in their GitHub repository	• Libraries and structures provided by the core team, e.g. d2 and bundled apps • External UI libraries • Many self-made reusable components and libraries • "Seed app" – standardized template app • Made resources that are also used by other app developers

5.4 Challenges with App Development

The HISP India developers had several times described time pressure, with short deadlines and high expectations from the client. In addition to developing applications, the developers had many other additional tasks, such as making HTML reports, integrations towards other systems, customization of the core, and technical support for their clients and the implementers. In other words, the developers often had a lot of other things to do besides working on applications. The challenge of having several other tasks was also mentioned by the HISP Tanzania team, who stated that they did "*Some work on development and innovations, but at the same time being an implementer*", and expressed that the many tasks and short deadlines were challenging.

Over time, the HISP India developers had made several custom applications. One developer expressed that "*custom apps are uncontrollable for us now*". The HISP India developers had developed many custom applications, to the point that maintaining all of the custom applications, had become one of their bigger tasks, as well as a major challenge. Also, new requirements for the existing custom applications kept coming from the clients. Hence, a lot of the application related work was not making new applications, but rather making changes to and maintaining the custom applications. Maintenance was also, as mentioned, a challenge for the HISP Tanzania developers. In contrast to the HISP India developers, HISP Tanzania had taken measures to control the maintenance work, and the long-term cost of maintaining a custom application would always be taken into account when they started working on a solution. By developing libraries of components and using these when developing applications, the HISP Tanzania developers would only have to maintain the components in the library and they would be updated within the applications that used those components. HISP India would often reuse components by copying and pasting code from their repository into their applications. A consequence of this approach was that whenever a component needed to be maintained or changed, they would have to do so in all the applications that had used that part of code. Hence, even though HISP Tanzania and HISP India both deemed it challenging to maintain the custom applications, HISP Tanzania had the capacity to make solutions that would reduce the long-term maintenance work needed.

Another challenge HISP India had was that it was very hard to upgrade the DHIS2 instances. The client did often not see the value of doing so, and even if they got the client agreed that it was necessary, it could take up to 1.5 years just to get the approval and funding to do so due to bureaucracy. If they did get the approval and funding necessary to upgrade, they would then have to do the necessary changes in the new version of the implementation, such as redoing the customization of the core, which was often required due to security requirements from the client. Sometimes, configurations would also have to be re-done, and the custom applications would also have to be maintained since upgrades often came with some changes to the core. Upgrading the DHIS2 instance had widespread consequences. Both the two implementation-specialist groups expressed that a challenge with upgrading was that by doing so, the UI in for example bundled applications tended to change a bit. Changes in the UI could become quite expensive, as this would often result in the need for retraining many of the end-users.

HISP India also expressed that their old DHIS2 versions often posed challenges for them. A consequence of not upgrading was that that they did not get access to new features they knew existed in newer versions. This meant that if their clients wanted these features, they would have to do so by doing custom changes in some of their applications or provide new applications for the client. In other words, they would have to develop something custom of which the core team had already developed and provided.

The last major challenge for HISP India was that there was a quite high turnover among the developers. Between our first and second trips, four developers had quit, and four new had been hired, meaning only two of the developers had maintained. New developers received little training and had few resources at their disposal to learn. Since the developers worked alone, it was therefore hard to transfer the knowledge of the experienced developers to the new ones.

Even though the two implementation-specialist groups did not necessarily have all the same challenges, the challenges will be summed up together. In total, there are four main challenges in relation to application development in the two implementation-specialist groups:

- Time pressure and many other tasks.
- A lot of work with existing custom applications due to maintenance and changes needed because of new requirements.
- Many instances with old versions. Costly to upgrade software instances to get new features. Hence, have to custom develop some of the same features, even though the features are already provided by the core developers in newer versions.
- High threshold for learning new skills, combined with little training for new developers and high turnover of developers.

5.5 Core Developers Work to Strengthen the Design Infrastructure around App Development

This section will present the new resources developed by the core team aimed at strengthening and supporting application development. The background for developing the new resources; the Design System and the App platform, was to give developers a framework that they can use to easier create and maintain applications for DHIS2. The new resources have a dual aim, which is to strengthen the internal development process of the core team and to strengthen external development processes for others such as the implementation-specialist groups. The following subsections will present how the new resources, the DHIS2 Design System and the App platform, evolved.

5.5.1 The DHIS2 Design System

The DHIS2 Design System is an initiative by the core team, to make it easier for internal and external developers to create a consistent UI throughout their platform and applications, as well as making it easier for them to develop and maintain their software. As mentioned, before the implementation of the Design System, DHIS2 had a library called d2-ui. The d2-ui library was written in pure JavaScript, which meant that the components could be imported into any JavaScript file, regardless of which framework that was used. The development of the new Design System started late in 2018. It started as a simple UI library, consisting of mainly freestanding elements, but also a couple of integrated components (such as the DHIS2 header bar). The UI library was based on the d2-ui and was developed to replace it. In addition, the core team also hired a visual designer as a consultant, to get the help of an UI expert when developing the new resources. The visual designer ensured that the UI was modern looking, consistent, and user friendly. The core developers did the technical implementation of the resources.

The UI components are created with React, so they could only be used by React applications. To structure the different types of components clearer, the core team created two libraries; DHIS2/ui-core and DHIS2/ui-widgets. The DHIS2/ui-core consists of all the freestanding components which have no dependencies to DHIS2. The DHIS2/ui-widgets consist of more complex components that have dependencies and need to be integrated with the DHIS2 instance.

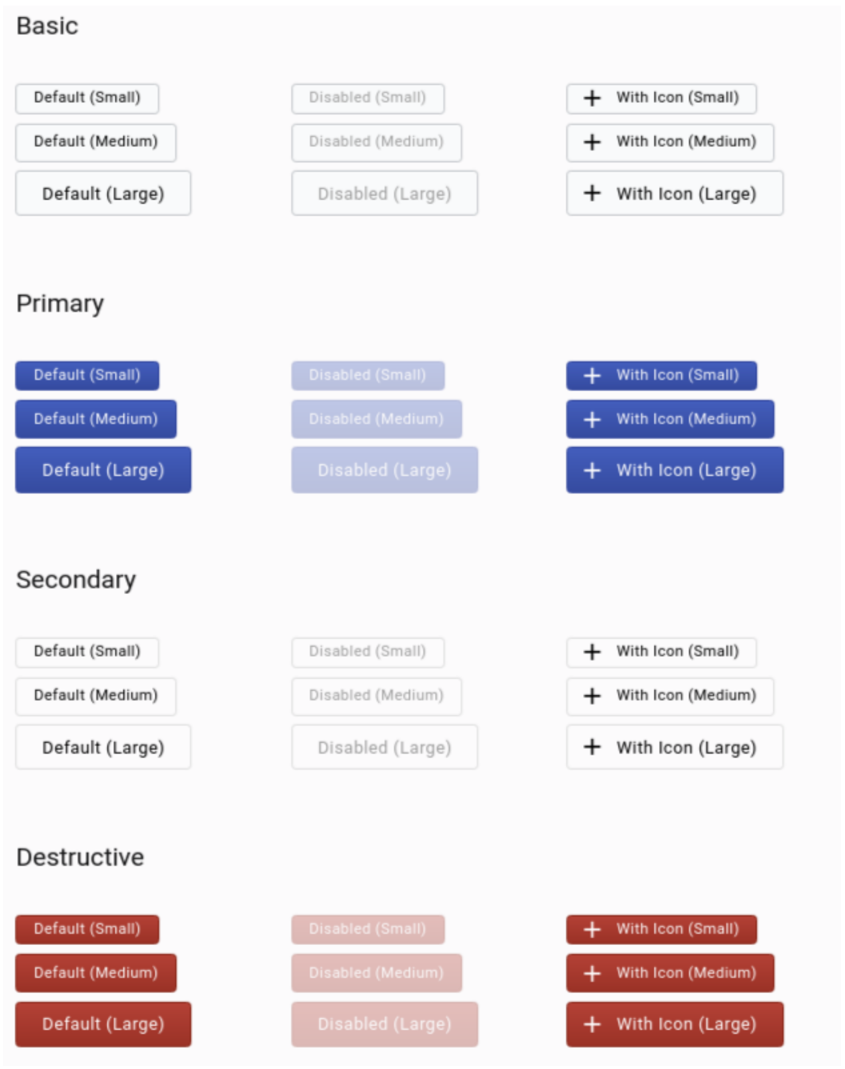
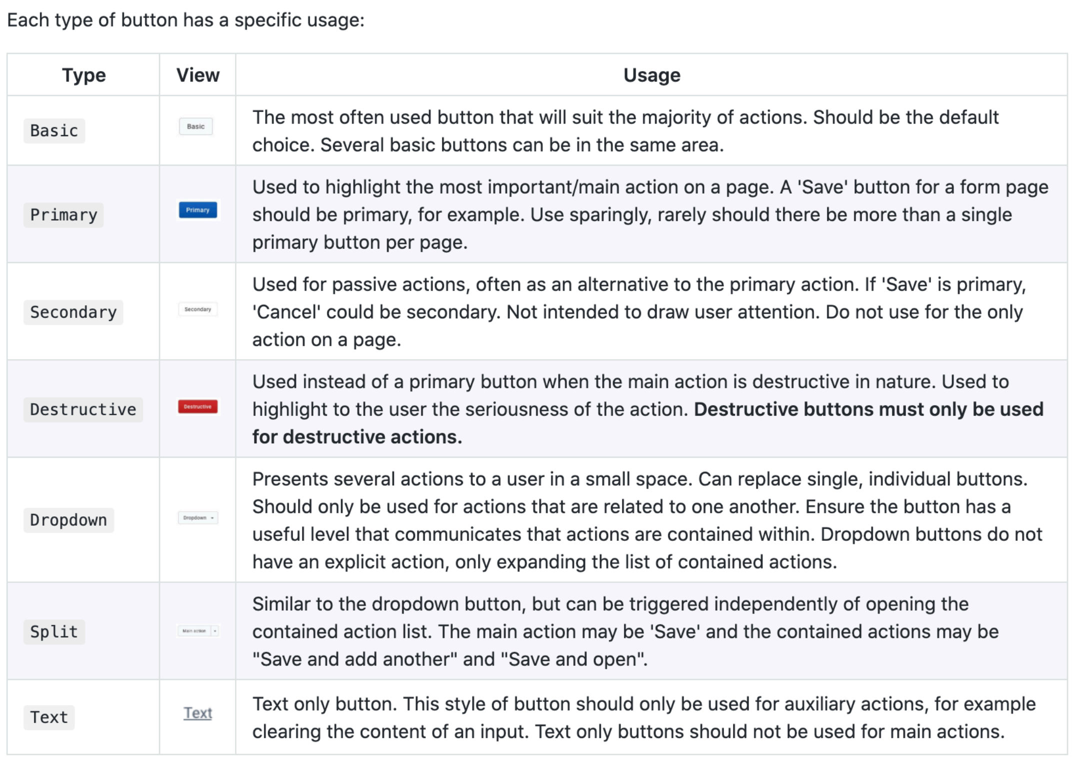


Figure 14: Some of the components in the first version of the DHIS2 UI library.

In April 2019, the Design System changed quite a bit. The components were moved to a website that is a development environment and playground for UI components. The website allows you to view a component library, and interactively develop and test components. This was done to make it easier to provide documentation and example code for each component.

More elements were also added to the Design System, in addition to the UI library, *standards* for when and how these components are to be used, and *design principles* were added. All the new information and resources were added to GitHub, where each component has a page, showing the standards and documentation for that specific component. On the top of the components page, there are links to the component's design specifications and a working example of the component. The information provided is an explanation of what the component is used for, and how to use it when developing the UI for an application. It also provides the different options you have to *configure* the component, and which states the component can have. Lastly, there is an example showing an actual use of the component in DHIS2.

Following the example of the buttons, the page provides a list of different types of buttons as well as when the different types should be used, as illustrated in Figure 15.



Each type of button has a specific usage:

Type	View	Usage
`Basic`	Basic	The most often used button that will suit the majority of actions. Should be the default choice. Several basic buttons can be in the same area.
`Primary`	Primary	Used to highlight the most important/main action on a page. A 'Save' button for a form page should be primary, for example. Use sparingly, rarely should there be more than a single primary button per page.
`Secondary`	Secondary	Used for passive actions, often as an alternative to the primary action. If 'Save' is primary, 'Cancel' could be secondary. Not intended to draw user attention. Do not use for the only action on a page.
`Destructive`	Destructive	Used instead of a primary button when the main action is destructive in nature. Used to highlight to the user the seriousness of the action. **Destructive buttons must only be used for destructive actions.**
`Dropdown`	Dropdown	Presents several actions to a user in a small space. Can replace single, individual buttons. Should only be used for actions that are related to one another. Ensure the button has a useful level that communicates that actions are contained within. Dropdown buttons do not have an explicit action, only expanding the list of contained actions.
`Split`	Main action	Similar to the dropdown button, but can be triggered independently of opening the contained action list. The main action may be 'Save' and the contained actions may be "Save and add another" and "Save and open".
`Text`	Text	Text only button. This style of button should only be used for auxiliary actions, for example clearing the content of an input. Text only buttons should not be used for main actions.

Figure 15: Usage of the different types of buttons in the DHIS2 Design System.

Further, the documentation explains the options when using a button, such as the option to add icons to it, or how to change the size, choosing from the three predefined options of small, medium, or large. Documentation of the buttons *state* options (e.g. for buttons you can choose to set the button as disabled) is also provided, such as how to disable the button and when this should be done.

The newly added design *principles* consist of five focus areas. These provide the developer with information on which typography that is to be used, which color spectrums to confine to, and a scale of spacing between elements that should be used as well as how the elements should be aligned and stacked. The design principles also explain how to communicate content and write understandable messages for the end-user with examples of "good" and "bad" communication. Lastly, it also provides guidelines for how to use icons and suggests using the open-source library *Material Design Icon Library* for getting the icons, as this library is the one the core team uses in their bundled applications.

5.5.1 The DHIS2 App platform

During the summer of 2019, the Core team initiated the development of what they refer to as an *App platform.* This is a unified application architecture and build. In other words, it enables the developer to easily create a sort of template application, consisting of all the common parts that are shared in about every application for DHIS2. The purpose of the app platform is to standardize and simplify application development for DHIS2, as well as making it easier to maintain the applications. One of the core developers explained that an app developer should only be concerned with the applications "*secret sauce*", that is the unique aspects of the specific application, and the rest of the application's structure should be provided by the app-platform. The UI components used in the App platform was of course from the Design System, and the Design System was an integrated part of the App platform. Figure 16 illustrates which parts of a DHIS2 application that is shared for all applications and that are provided by the App platform, as well as the applications "*secret sauce*" that is parts that are custom in the specific application.

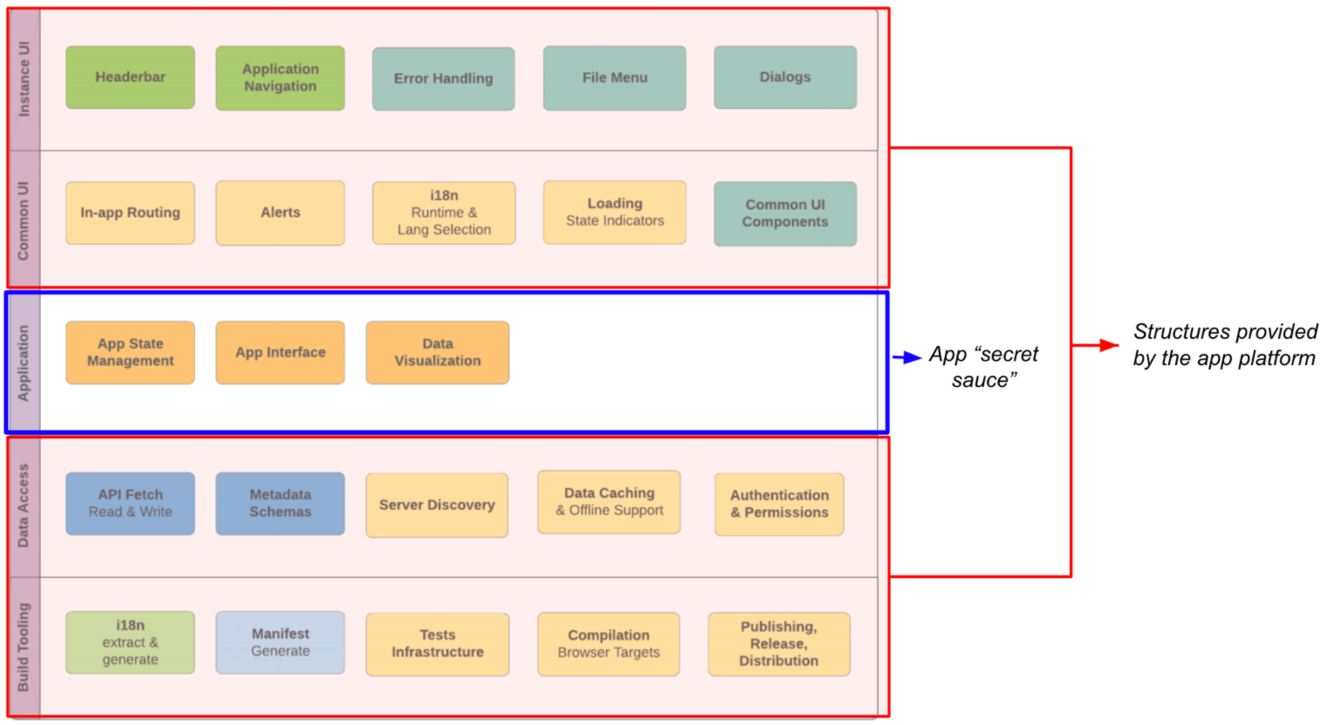


Figure 16: Illustrates structures provided by the app platform and the applications "secret sauce". Based on a figure provided by the core developers.

Developing an application with the App platform is not much different than starting from a forked application. The developer starts with a lot of existing structures, and do the changes required for the specific implementation. However, the structures provided by the App platform are maintained by the core team, and changes are updated within all the applications made with the App platform without having to go into each application and do the changes. In forked applications, the code is copied, thus the developer has to maintain all of the code in every fork made.

Figure 17 illustrates how the Design System has evolved, from first consisting of UI components to the second version where guidelines for using the components and design principles were added. In the third version, the Design System itself did not change, however, the App platform was developed, and the Design System became an integrated part of the App platform. However, it must be clarified that it is still possible to develop applications using the components from the Design System without using the App platform, and vice versa. It is also possible to import other external libraries into an application made using the App platform. Hence, the Design System and the App platform are two different resources, fulfilling each other. The idea is that these two resources combined can offer all the building blocks needed when developing an application for DHIS2.

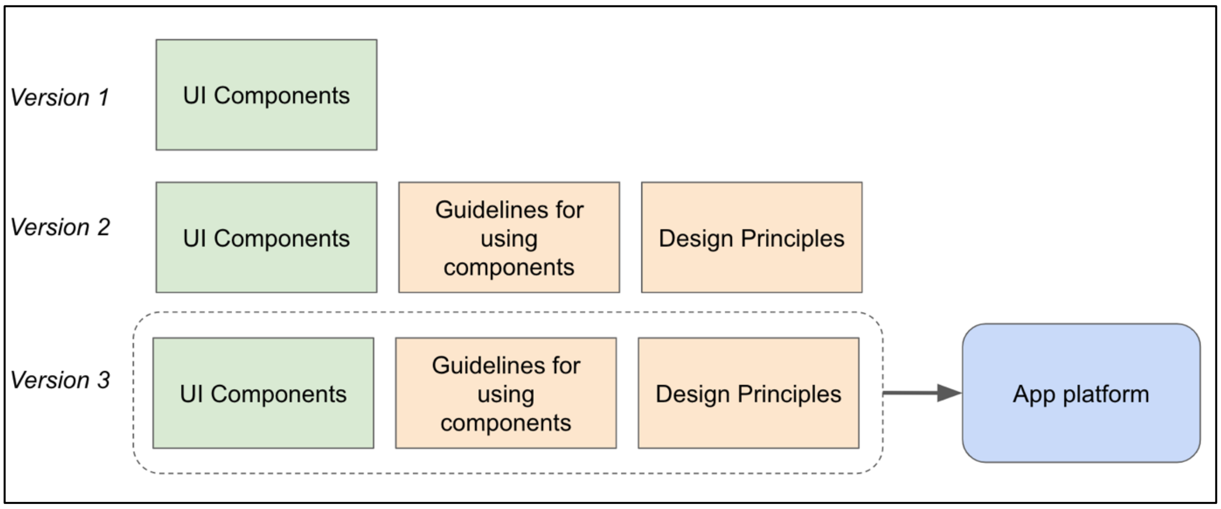


Figure 17: Evolution of DHIS2 Design System.

6. Analysis

The aim of the analysis to get a better understanding of how the core teams newly developed resources can strengthen the implementation-specialist groups during the implementation-level design of custom applications. I will analyze how the new resources can fit into the context of HISP India and HISP Tanzania. I will also present the challenges the implementation-specialist groups face regarding utilizing the new resources in the DHIS2 design infrastructure.

6.1 How the New Resources Fit the Practices and Challenges

As previously mentioned, one of the core team's aim with the new resources in the DHIS2 design infrastructure is to strengthening external development processes. This section will look into how the new resources could fit in with the practices and challenges that HISP India and HISP Tanzania face, and if it can support the implementation-specialist groups' developers in app development. I will first start with what the implementation-specialist groups expressed as beneficial and desirable traits with the new resources. After this, I will go through the various challenges the implementation-specialist groups face in utilizing the new resources. Through the challenges, it will become clearer how the new resources fit or do not fit, with the implementation-specialist groups existing skills, structures, and practices, and how it can be addressed.

6.1.1 Benefits for the Implementation-specialist Groups

Two main benefits, or desirable traits, that the implementation-specialist groups wanted with the new resources, were identified. These were related to both the implementation-specialists groups need for reducing the long-term cost of developing applications, and a desire for a standardized UI. However, as maintenance cost often was perceived as a barrier to making new applications and most clients did not really care about the UI, the implementation-specialist group put most emphasis on the former benefit.

Maintenance

A benefit of the App platform is that it moves much of the responsibility of maintenance from the implementation-specialist groups to the Core team. This was a highly desirable trait for the two implementation-specialist groups, as both mentioned that maintenance work on custom applications was an activity that consumed a lot of time and resources from the teams. Maintenance was one of the main considerations when deciding to make a custom application. It was also one of the reasons for why they would often go to great lengths to manage to configure a solution, rather than develop custom applications. In other words, the new resources do cover a need that the two implementation-specialist groups have.

Standardized UI

The HISP Tanzania developers expressed that they were considering making a solution to better help them standardized the UI they used when developing applications. The developers said it was the next resource needed to strengthen their application development. When first presented with the DHIS2 Design System, one of the developers stated "*Now, we need standardizations. Perhaps this is what we need*". The HISP India developers also expressed that a standardized UI would be useful for them, as the team consisted of back-end developers who were not experts on UI. In other words, the team lacked the capacity of a visual designer, a gap that could be filled by using the Design System.

6.1.2 Challenges for the Implementation-specialist Groups

The Developers' Skillset

The reason why HISP India initially had chosen to use the Angular framework, was because many of the bundled applications developed by the core team had been in Angular. When they customized these applications, they continued working with Angular. Several of the developers expressed that Angular was preferred, as React was deemed a complex framework, and they were already very comfortable with using Angular. An often-mentioned challenge with the applications made is the issue with *cache*[1]. One developer said that it was perhaps one of their biggest challenges with custom applications and that it was easier to

[1] Cache, or here browser cache, is a quick, temporary storage area. It stores temporary internet files for faster viewing of the website in the future.

handle the cache in Angular, but complicated to do in React. They also mentioned that it was challenging to learn new frameworks, as they already were working under time pressure on existing tasks. Some of the developers in the HISP Tanzania team had started learning React and were considering transitioning over to it in the future. However, they did not like that they were forced to use React. As one developer said, "*we are trying to find our own way*". They were frustrated over the fact that the Design System had to be framework-specific, as the components in the previous d2-ui had been possible to use in any framework. The HISP Tanzania developers expressed that they would probably "*try to incorporate React components into Angular*", as Angular was their preferred framework for application development. Summed up, the developers in the two Implementation-specialist groups prefer to work with a framework they already know. They either did not want to use React, or they did know how to use it.

Existing Custom Applications

The HISP India developers explained that if they were to start on a completely new application, they would consider using the App platform and the Design System. However, most of their existing custom applications were developed in Angular. One of the HISP India developers expressed that they do want to align with the core team in terms of for example the framework used for developing applications and the design infrastructure provided globally. However, now that about all of their applications were in Angular, it would take a lot of time rewriting them in React, which was time they did not have. The HISP Tanzania shared the same concerns and expressed that rewriting their custom applications was out of the question as it would take too much time.

Other Overlapping Resources

The two implementation-specialist groups had developed many resources for themselves, many of which provided overlapping functionality with the new resources from the core team. Both implementation-specialist groups utilized third-party UI component libraries, which was supported by Angular. The HISP Tanzania developers expressed that if the Design System had provided more components that consisted of both UI and functionality together, this would be a valuable resource, however for pure UI components they already had other UI libraries they could use. The HISP Tanzania team had also developed the "Seed app", which provided a lot of the same features as the App platform. In other words, much of what the new resources provided was already somewhat covered by local resources.

Missing out on Resources

As mentioned in section 6.2, the two implementation-specialist groups have developed many resources to support their effort in developing custom applications for DHIS2. Most of these resources are specific to Angular. A consequence of the local resources being in Angular is that the developers would not be able to utilize their own resources if they were using the new resources provided by the core team. For example, it would not be possible for the HISP Tanzania developers to start with their Seed app and use the UI components from the Design System when making an application, as the frameworks were incompatible. Many of the resources the developers at the two implementation-specialist groups used were related to functionality, rather than UI. Even though the App platform provides a lot of functionality, the implementation-specialist groups had also made libraries to support more case-specific functionality. An example of this was an interactive visualizer developed by the HISP Tanzania developers, which they had made because DHIS2 only provided static visualizers. In other words, due to the frameworks being different, many of the resources developed globally and locally are incompatible with each other. By utilizing the new resources, they would miss out on many of their own resources' features. This would leave the development of applications with an unfilled gap, which they would need new solutions or resources to fill.

This challenge was particularly evident for the HISP Tanzania developers. They had not only developed a range of resources for themselves, but many of their resources were also developed for or used by external implementation-specialist groups and app developers for DHIS2. If they were to discontinue the support for their libraries, it would not only affect their custom applications, but also many other custom applications.

Skepticism toward the New Resources

The developers at HISP India were in general a bit frustrated with the communication from the core team regarding the changes and new resources that were coming. The HISP India developers were concerned that if they started using React, the core team might later switch over to something else like the core team had done when moving many of the bundled applications from Angular to React. They were questioning if the core team now was actually committed to the Design System and App platform, as they now would discontinue support for the d2-ui, a resource which they had utilized quite a lot. They were also concerned about the Design System and App platform's reliability, and whether it had been thoroughly tested before it was released.

Older Versions of DHIS2

As mentioned, many of the DHIS2 instances HISP India have implemented and maintain are quite old versions of DHIS2. Due to both budgets and bureaucracy, it could take the HISP India team up to 1.5 years to upgrade to a newer version. They first had to get approval and funding, and then do the necessary changes in the new version of the implementation, such as customizing parts of the core to get a required security certificate. As the core team releases new versions every half year and only supports the three last versions, they would "fall behind" very fast. Another reason both implementation-specialist groups expressed was a challenge with upgrading, was that by doing so the UI in for example bundled applications tended to change a bit. Changes in the UI could become quite expensive, as this would often result in the need for retraining many of the end-users. These factors made upgrading both challenging, and sometimes also undesirable.

As the App platform and the Design System was recently developed, it had been developed with the newest version of DHIS2 in mind. This meant that even if the HISP India developers wanted to use the App platform and the Design System, it would not even be possible in many of the instances they were working on, as the versions were too old.

Clients Do Not See the Value

Another challenge was that both upgrading and rewriting applications was decided by the client. The HISP India team could recommend and try to persuade the client, but in the end, it was the client's decision. As mentioned in the last section, upgrading could often have widespread consequences, such as the need for retraining users. This would be costly for the client, who often did not see the need for upgrading in the first place. For the HISP India developers to utilize the new resources, they would not only have to rewrite the many custom applications and learn React framework, they would also have to upgrade their versions. Investing in upgrading the version would be necessary to utilize the new resources, as the new resources were not supported by the older versions. Both upgrading the version and rewriting existing custom applications would be a very costly affair for the client. As most of their clients have limited technical knowledge, they would not see the value of doing this transition and would most likely not be willing to pay for it to be done.

6.1.3 Summary of Analysis

As mentioned in earlier sections, the Design System and the App platform have some traits that the two implementation-specialist groups desire and need. However, as presented, there are also many challenges related to the new resources fit within the two implementation-specialist groups. The challenges are summarized in Table 5.

Table 5: Challenges with the new resources' fit with the implementation-specialist groups.

Challenge	**HISP India**	**HISP Tanzania**
Developers skillset	Use and know Angular framework. Do not have the time or resources to invest in learning React.	Use and know Angular framework. Do not want to be "forced" to use other frameworks.
Existing apps	About all of their existing apps are developed in Angular. To utilize the new resources all applications would need to be rewritten.	About all of their existing apps are developed in Angular. To utilize the new resources all applications would need to be rewritten.
Overlapping resources	Many of the resources they already know and use, partly overlap with the new resources. Do not have the time to learn how to use the new resources, when they already have something they know.	Many of the resources they already know and use, partly overlap with the new resources. Prefer to use something they already know.
Missing out on features because the new resources are incompatible with their own resources	Would not be able to utilize the resources they currently do, due to the frameworks being incompatible.	Would not be able to utilize many of their own components and libraries if they are using the new resources, due to the frameworks being incompatible.
The versions of the DHIS2 instances	Have many implementations of DHIS2 which's versions are not compatible with the new resources.	-
Clients do not see the value	The clients often have limited technical knowledge and do not see the value of converting existing apps over to the App platform and Design System.	-

7. Discussion

This chapter will start with an overview of the key findings of the analysis. Following this, I will present what these findings mean for the DHIS2 as a design infrastructure, and I will propose the terms *local design infrastructure* and *global design infrastructure* to elaborate on the dynamics between implementation-level design, the generic-level design, and the new boundary resources. These findings will then be discussed with related literature. Through discussing the empirical findings and related literature, I will answer the research question;

What affects a platform boundary resource's fit in an implementation-level context?

Following this, I will present the implications the research has in relation to theory, as well as limitations of the research. In the next chapter, I will propose practical contributions in form of some considerations enterprise software platform vendors should take into account when developing new boundary resources.

The empirical findings have shown that there are several challenges related to the DHIS2's new resources fit in the two implementation-specialist groups. One of the key obstacles both the implementation-specialist groups face concerning app development is the buildup of maintenance obligations and the high cost that follows. This comes in addition to the already high cost of developing applications. However, the two implementation-specialist groups have dealt with the challenge of cost in two different ways. HISP Tanzania has taken some measures to reduce the long-term cost of maintenance and also reduce the short-term cost of app development by developing their own resources to support this. The resources developed and used by HISP India address their need for reducing the short-term cost of developing applications, but not the long-term cost of maintaining them. As mentioned, HISP India described their custom applications as becoming "uncontrollable", due to the increasing amount of custom applications and the maintenance obligations that follow. HISP Tanzania does not face the same pressing matter, even though they have developed a higher number of applications. There can be many reasons for this, such as differences in funding, different scope and scale of projects, and so on. However, parts of the differences are also due to the differences in the practices and structures. Despite these differences, both the implementation-specialist groups face many of the same challenges related to the new boundary resources.

The findings highlight some challenges that affect the fit of DHIS2's new boundary resources at the implementation level. Table 6 illustrates a more generalized summary of the challenges with the new boundary resources in the context of the implementation-specialist groups, which was found in the analysis.

Table 6: Challenges that affect the fit of new boundary resources at the implementation-level.

Fit related to	**Challenge**	**Explanation**
Capacity	Developers existing skillset	Existing skillset not compatible with the new resources.
	Resources and capacity to acquire new skills	Do not have the resources and capacity to acquire the new skills needed to utilize the new resources.
Existing structures and resources	Existing app development framework used in implementations	Existing implementations of apps are not compatible with resources in the design infrastructure, such as the apps having different app development framework that is incompatible with the new boundary resources framework.
	Existing implemented versions of the enterprise software	The new boundary resources are not supported by older versions of the software, and as many of the implemented instances have old software versions the implementations are incompatible with the new boundary resources.
	Existing app development resources	Resources utilized in current practices are not compatible with the new resources.

The findings have shown that introducing new boundary resources to an implementation-level context is not a straightforward process. The challenges relate both to the technological and social aspects. Technological in relation to existing structures, such as existing custom apps and the frameworks used, the versions of the enterprise software, and app development resources. It also relates to social aspects, such as the developer's skillset and their possibility of acquiring new skills. I will elaborate on each of these challenges in relation to literature and in section 7.3. Before doing so, I will elaborate on what this means for the DHIS2 *Design infrastructure.*

7.1 Enterprise Software and Boundary Resources as a Design Infrastructure

The challenges identified relate to the structures and people's capacity that exist implementation-level context and practices during implementation-level design. These are all a part of the DHIS2 design infrastructure, as the design infrastructures cover all activities and arrangements surrounding the system (Li & Nielsen, 2019). However, it can be useful to distinguish between where the resources and design that may enable design, comes from within the design infrastructure to understand the dynamics. Elaborating further on the concept proposed by Li and Nielsen (2019) on design infrastructures, it can be argued that the two implementation-specialist groups have over time developed their own *local design infrastructures*, consisting of many resources, structures, and practices that they draw upon in implementation-level design of the enterprise software DHIS2. It can thus be said that implementation-level design is a process that happens within the local design infrastructure and that it is a process that utilizes the resources from within the local design infrastructure. The local design infrastructure can be seen as a part of the *global design infrastructure.* The processes of generic-level design that the core team do, such as development of new boundary resources, takes place within the global design infrastructure. It can be argued that the local design infrastructure has been developed to compensate for what is lacking in what is provided by the core team.

Further, it can also be argued that the two implementation-specialist groups have different practices, capacity, and resources, and hence, different local design infrastructures. The findings also show that the local design infrastructures must be seen in relation to both the global design infrastructure as well as other local design infrastructures. As mentioned, the local design infrastructure draws on practices and resources from many other sources besides what is provided by the global design infrastructure, such as resources from other local design infrastructures, structures and resources they have themselves developed, and also external resources provided by parties not related to the enterprise software, such as generic UI libraries. However, the findings also show that parts of the local design infrastructure can "move out" from the specific local design infrastructure into the more widely accessible global design infrastructure. For example, HISP Tanzania had developed many generic applications that had been published to the DHIS2 online store, as mentioned in Section 5.2. These applications are developed within a local design infrastructure, but the global design infrastructure facilitates and supports the sharing of the applications. Hence, these applications become more accessible to everyone within the global design infrastructure. Another example is how the HISP India team organized academies held for other DHIS2 complementors and implementation-specialist groups to strengthen capacity, where the academies were in cooperation with and facilitated by the core team. Besides this, the implementation-specialist groups were also sharing resources from their local design infrastructure more directly to other local design infrastructures, such as sharing existing custom applications that had not been published in the online store. Even though the local design infrastructure is a part of the global design infrastructure and have many shared resources, practices, and structures, a differentiation between the two can be useful to describe the dynamics of how resources, practices, capacity, structures, and other arrangements may influence each other, be shared, or how they co-exist and respond differently when the local design infrastructures do not fit with resources and structures provided by the core team. The relation between the global and the local design infrastructures can thus be illustrated as seen in Figure 18.

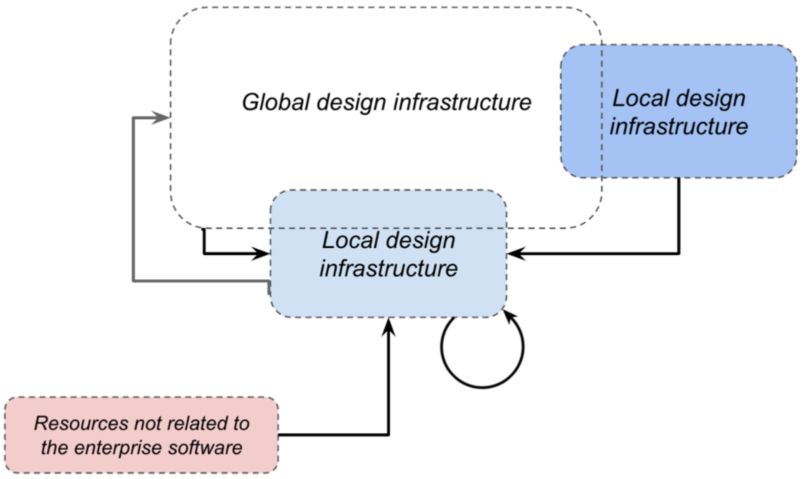

Global design infrastructure
Local design infrastructure
Local design infrastructure
Resources not related to the enterprise software

Figure 18: Dynamics between the global and local design infrastructures.

Therefore, in order to properly understand how the new boundary resources' fit on implementation-level, it must be seen in relation to both the DHIS2 global design infrastructure as a whole and the specific local design infrastructure, as illustrated in Figure 19. As other app developers for DHIS2 might have other local design infrastructures, the DHIS2 new boundary resources might have a better fit in other organizations, and it might also pose other challenges.

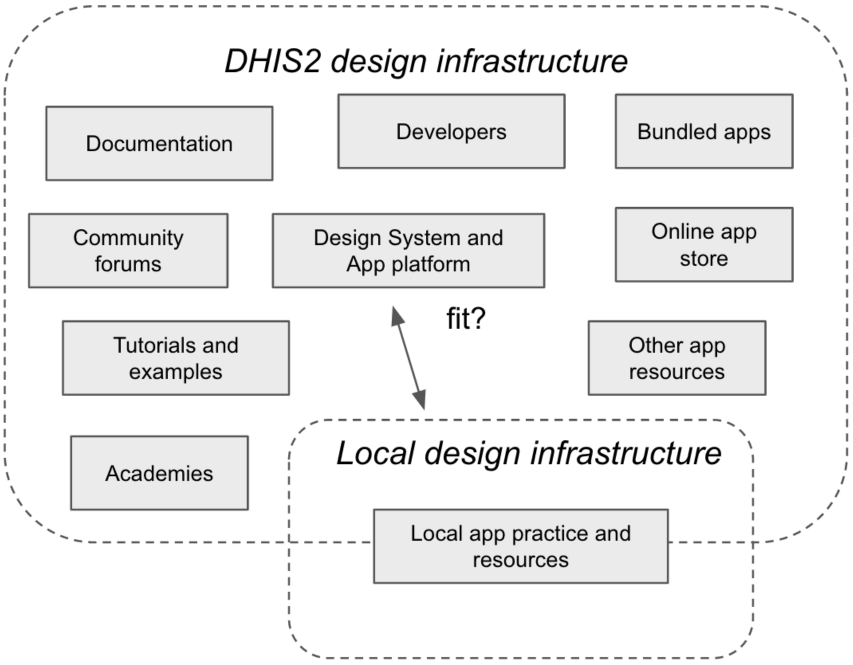


Figure 19: Relation between the Global design infrastructure and the Local design infrastructure.

7.2 Challenges of the Boundary Resources' fit with the Local Design Infrastructure

Many of the challenges found associated with the introduction of new boundary resources can be ascribed to the boundary resources being incompatible with the local design infrastructure. I will now discuss how these challenges relate to other literature.

7.2.1 Fit Related to Capacity

It was found that many of the developers in implementation-specialist groups did not have the skills to utilize the new boundary resources, and it was found challenging for the developers to get the resources and capacity needed to acquire these skills. Literature has argued that platform architectures inherently pose "knowledge boundaries" as that platform complementors must know how to access and extend the platform (Foerderer et al., 2019), the empirical findings supports this claim. The empirical findings suggest that the knowledge boundary can even extend beyond this scope. The two implementation-specialist groups did have the skills and capacity to develop applications for the platform, which they had already done so many times before. However, they did not have the skills and capacity to utilize the new boundary resources, that were provided to better support the development of applications. In other words, knowledge of the software itself is not necessarily enough to efficiently extend the software, resources aimed at supporting these activities can also pose knowledge boundaries. The findings show that the implementation-specialist groups cannot utilize the new boundary resources unless they acquire new skills that can make it possible for them to leverage the new boundary resources. Acquiring new programming skills can take a lot of time. This would be a burden for the implementation-specialist groups, as time spent on acquiring the new skills could have been spent on other, and perhaps more pressing, tasks. This can be a time-intensive task, and thus, comes with a quite high cost. Developing custom applications has previously been identified in literature within implementation-level design as a time- and competence-intensive task (Li & Nielsen, 2019). However, elaborating further on these findings, a related challenge can be the cost in the time needed to acquire the competence necessary to utilize the platform boundary resources. The empirical data indicates that more competence, or advance competence, is needed within the implementation-specialist groups for them to be able to utilize the new boundary resources. Hence, the more competence needed, the greater the barrier seems to be.

The findings also show that knowledge boundaries must not only be seen in relation to the boundary resources and the complementors capacity, but also to the complementors ability to acquire the more fundamental skills needed to utilize technology that supports the boundary resources. For the resources to support app development in this context, these knowledge barriers, or misalignment in the organizations' capacity and skills to utilize the resources, must be addressed (Foerderer et al., 2019). It has been suggested that more interactions between the platform vendor and the complementors is an approach to build competence around app development and to address the knowledge boundaries (Bianco et al., 2014; Foerderer et al., 2019). There are several approaches that has been found viable and beneficial to address knowledge boundaries, but that are not currently present in the existing DHIS2 design infrastructure or that can be strengthened in the DHIS2 design infrastructure, such as hackathons, live chats or help desks, example code, hosted developer sandbox (developer testing environments), and alignment workshops (Bianco et al., 2014; Foerderer et al., 2019). The empirical findings have shown that the implementation-specialist groups have quite good insight into many of the domains to which they create solutions as well as the technical aspects of the enterprise software. Decreasing the knowledge boundaries could also be an efficient approach to bridging the needed context-specific information with the generic enterprise software, and thus to create more feasible and desirable solutions (von Hippel, 2005). However, even if the knowledge boundaries are addressed, the boundary resources fit is not straightforward due to the existing implementations and resources, which will be elaborated on in the following section.

7.2.2 Fit Related to Structure

Existing Implementations

The implementation-specialist groups existing implementations of applications are not compatible with the app framework the new boundary resources supported. This posed a challenge for the implementation-specialist groups as they would not be able to utilize the new boundary resources without putting in a lot of effort and rewriting their existing apps over to another app development framework. The implementation-specialist groups argued that they had adapted their solutions to previously provided boundary resources and bundled applications. It was the core development team that changed the app framework, and thus, caused this challenging misalignment.

Previous research has argued that the platform vendors balance flexibility and stability by providing a stable platform core, and flexibility by allowing complementors to extend the software (de Reuver et al., 2018; Tilson et al., 2010). It has been argued that the boundary resources enable this flexibility and stability (Tilson et al., 2010). However, other researchers have stressed the importance of stability in boundary resources (Bianco et al., 2014). The findings in this research strengthens these findings, as the case has shown that the implementation-specialist groups do not only adapt to the stable core, but also to the boundary resources provided by the platform vendor. It has shown that if the new boundary resources are too different from previously provided resources and structures, it might not be viable resources for the implementation-specialist groups. Thus, to address this challenge, the platform vendor should aim at providing stable boundary resources in addition to a stable core, to serve as a solid foundation for growth (Bianco et al., 2014).

Existing Resources

As previously mentioned, the implementation-specialist groups have developed and evolved their own resources over time, through a process of self-sourcing (Ghazawneh & Henfridsson, 2013). Research has had a limited focus on the “receiving” end of boundary resources (Eaton et al., 2015). This case has highlighted the practices and challenges the platforms new boundary resources pose for the implementation-specialists group. It has shown that the complementors are not passive receiving parties of the boundary resources provided by the core team, but rather supports previous research that has argued that complementors actively assess and shape the resources (Eaton et al., 2015). Not only are the boundary resources enabling creations of applications to the software, but they are also utilized in the implementation-level context to develop new resources. Ghazawneh and Henfridsson (2013) argue that this *self-resourcing* is a response to the perceived limitations in the existing boundary resources. The findings in this case suggests that there are several other mechanisms and reasons for why the implementation-specialist groups develop and use their own resources besides the perceived limitations in the existing boundary resources. Part of the reasons is that it is more efficient in short-term for complementors to use resources that they have the skills and capacity to use, as the new resources pose knowledge boundaries that must be addressed before they can be utilized (Foerderer et al., 2019). Another reason is if the boundary resources are not compatible with the established structures and practices, which makes the resources less desirable. The implementation-specialist groups had developed a range of resources to support their own app development practices in addition to the resources based on the platform’s boundary resources. The implementation-specialist groups had also developed several app resources based on other sources, such as the platforms bundled applications, external UI component libraries, and existing applications from other complementors. The reasons for, and sources of, self-sourcing are summarized in Table 7.

Table 7: Reasons for self-sourcing, and sources of self-sourcing found in the implementation-specialist groups.

Reasons for self-resourcing	• Perceived limitations in the existing boundary resources (Ghazawneh & Henfridsson, 2013). • Easier to use self-developed resources than the boundary resources. • Boundary resources are not compatible with existing structures, practices, and skills. • New boundary resources being incompatible to previously provided boundary resources.
Sources of self-resourcing	• The DHIS2's previously provided boundary resources. • Existing structures in the design infrastructure, e.g. bundled applications. • Existing structures from other complementors to the platform, e.g. applications developed by third parties. • Their own existing structures e.g. previously developed custom applications. • Sources outside of the platforms design infrastructure, e.g. UI component libraries.

It has been argued that self-sourcing is often aimed at achieving goals that might be conflicting with the platform vendors goals, and thus limited by the boundary resources (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013). Both the DHIS2 core team and the implementation-specialist groups share a common overarching goal of strengthening health. Even though the implementation-specialist groups also do implementations for other domains, this is neither conflicting nor competing with the platform vendors goals, who are aimed at developing an enterprise software platform as a public good. The boundary resources provided by the core team are not aimed at controlling or limiting the platform complementors from doing any particular things. Thus, this case shows that self-sourcing also happens also in the context where the platform vendor and the complementors do not have conflicting or competing goals. This provides a basis to broaden the understanding of boundary resourcing, besides its relation to platform securing (Ghazawneh & Henfridsson, 2013).

Boundary resources and the concept of self-sourcing also provides a further understanding of the relationship and dynamics between the boundary resources provided by the platform vendor and the implementation-level context, as well as a better understanding of how the receiving end of boundary resources utilize the boundary resources and other resources to strengthen their internal development of applications. Thus, it also contributes to further the understanding of platform governance dynamics.

7.3 Contributions

The investigation of an enterprise software platform's boundary resources fit in an implementation-level context, has contributed to the research field in four ways. I have proposed the perspective of a *local design infrastructure* that is a part of the broader *global design infrastructure*, and that this differentiation is useful to understand the dynamics of what is going on between implementation-level contexts and the enterprise software platform vendor. The findings have contributed to a further understanding of how the platform architectures inherently pose knowledge boundaries (Foerderer et al., 2019). Furthermore, it has provided insight into the challenges posed by changing the technology around boundary resources, and the how platforms vendors must balance the paradoxical relationship between stability and flexibility, also in relation to the boundary resources (Bianco et al., 2014; Tilson et al., 2010). Lastly, the research has contributed to a nuanced understanding of reasons for why self-sourcing among platform complementors happen (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013), and that self-sourcing also happens in a context where the platform vendor and the complementors have non-conflicting goals.

7.4 Limitations

The research was carried out through a non-interventionist approach, as this gained insight into the articulated and observed experiences of the research participants. A more interventionist approach, such as Action Research, could have provided more practical insight. As the research was qualitative and with an interpretivist approach, my subjectivity as a researcher and a developer has also influenced the findings. However, without a background as a developer and technical insight, it would have been difficult to understand much of the findings and the context. Due to the limited scope and time, it was not possible either to study how the implementation-specialist groups had developed their own local design infrastructures. Other insights into app development practices could have been gained by following a development of specific application. However, due to the interest in the new boundary resources and how they could potentially support future app development, the focus was to get a more overall perspective on the practices and challenges in app development within the implementation-specialist groups.

8. Conclusion

Based on a two-year case study with research conducted within three different contexts, this thesis has discussed the technical and organizational challenges of introducing new boundary resources for an enterprise software platform into an implementation-level context.

The research has shown that the fit of boundary resources needs to be seen in a socio-technical perspective, as the resources need to fit with both technology (existing structures and existing resources), as well as in relation to the implementation-level contexts' capacity and existing practices. To understand and elaborate this, the perspective of *local design infrastructures* has been proposed to describe the various co-existing structures, resources, practices, and activities that happen within the *global design infrastructure*, as well as the dynamics between the implementation-level design and generic-level design.

The development of custom applications at the implementation-level has been found to be highly undesirable for the implementation-specialist groups due to the high short-term and long-term costs. Yet, when implementing enterprise software, custom applications are often needed to meet the needs and practices of the user. It is found that during the implementation of custom applications, the developers utilize a wide range of resources. Many of the resources are introduced to the global design infrastructure by the core team, but there are also many resources that have evolve from within the local design infrastructure. The local design infrastructures have evolved over many implementation projects, and over a long period. In other words, it can be argued that this local design infrastructure has been developed to bridge the gap where the global design infrastructure is insufficient. Due to this, it can be very challenging to introduce new boundary resources into this implementation-level context, especially if there are misfits between the boundary resources and the local design infrastructure. In other words, to make a global design infrastructure that supports implementation-level custom app development for enterprise software, the global design infrastructure and local design infrastructure should aim at aligning.

Practical contributions

When evolving a global design infrastructure, by for example adding new boundary resources, this should be based on, and be compatible with, what is already in the global design infrastructure. Too big changes can lead to complementors utilizing less of the resources provided by the generic-level, and rather expand their local design infrastructure with other resources that are compatible with their existing structures, practices, and implementations. Thus, the research provides a basis to elaborate on important aspects when introducing new platform resources into a design infrastructure' (for them to fit with the local implementation context):

- Technical resources should be as detached, or as loosely coupled, from dependencies as possible. E.g. aim at avoiding being framework specific.
- The resources in the design infrastructure should aim at being stable, yet with the possibility to evolve to cover new needs.
    - New resources should aim at being compatible with the existing resources, practices, and structures in the global design infrastructure.
- New resources should be coupled with documentation, tutorials, and/or examples that can equip people with the skills needed to utilize the resources. E.g. the DHIS2 App platform and the Design System should be coupled with tutorials on how to learn React.
- Resources should aim at removing as much of the cost (e.g. cost of development and maintenance) from the implementation-level and place these on the software enterprise platform vendor.
- Generic-level design should also aim at mechanisms within the global design infrastructure that allows sharing of the resources developed by the implementation-specialist groups.

These aspects can potentially better align the global design infrastructure with the local design infrastructures, and better strengthen and support the work of the implementation-specialist groups.

Future research

The findings in this thesis also provide a basis for several interesting research topics. Firstly, as this research illustrates how it is difficult to introduce new boundary resources into an implementation-level context, it would be interesting to elaborate on how implementation-level contexts actively adapt to boundary resources over time, and how they come to use them or create other solutions. For example, follow-up research in the future of this study would be interesting to see if and how the new boundary resources can become a better fit.

As this research shows, there is a misalignment between the global and the local design infrastructures. Future research should investigate further the dynamics between the global and the local design infrastructures and investigate how to better align these design infrastructures.

This research has also shown that there are differences in how the two implementation-specialist group's local design infrastructures have emerged and how they evolve. Even though the two groups have quite similar projects and clients, they have addressed the challenges related to app development differently. To understand the reasons for *why* and *how* the design infrastructures have become different, further research is needed.

Hopefully, this research can contribute to how enterprise software platform vendors develop resources intended to strengthen the work of the enterprise software complementors, and that this can make the complementors work easier.

9. References

Baldwin, C. Y., & Woodard, C. J. (2009). The Architecture of Platforms: A Unified View. In *Platforms, Markets and Innovation* (pp. 19–44). Edward Elgar Publishing. https://doi.org/10.4337/9781849803311

Bansler, J., & Havn, E. (1994). *Information systems development with generic systems.* 707–718.

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), 873–902. https://doi.org/10.25300/MISQ/2019/14214

Bianco, V. D., Myllärniemi, V., Komssi, M., & Raatikainen, M. (2014). The Role of Platform Boundary Resources in Software Ecosystems: A Case Study. *2014 IEEE/IFIP Conference on Software Architecture*, 11–20. https://doi.org/10.1109/WICSA.2014.41

Braa, J., Monteiro, E., & Sahay, S. (2004). Networks of Action: Sustainable Health Information Systems across Developing Countries. *MIS Quarterly*, *28*(3), 337–362. JSTOR. https://doi.org/10.2307/25148643

Braa, J., & Sahay, S. (2017). *The DHIS2 Open Source Software Platform: Evolution Over Time and Space*. The MIT Press. https://www.researchgate.net/publication/316619278_The_DHIS2_Open_Source_Software_Platform_Evolution_Over_Time_and_Space

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.-. https://doi.org/10.1109/HICSS.2001.927130

de Reuver, M., Sørensen, C., & Basole, R. C. (2018). The Digital Platform: A Research Agenda. *Journal of Information Technology*, *33*(2), 124–135. https://doi.org/10.1057/s41265-016-0033-3

Dean, J. C., Dr, C., & Vigder, M. R. (2002). *System Implementation Using Commercial Off-The-Shelf Software*.

DHIS2 design lab. (n.d.). Retrieved June 9, 2020, from https://www.mn.uio.no/ifi/english/research/networks/hisp/dhis2-design-lab/index.html

DHIS2 front page. (n.d.). Retrieved May 19, 2020, from https://www.dhis2.org/

Dittrich, Y. (2014). Software engineering beyond the project – Sustaining software ecosystems. *Information and Software Technology*, *56*(11), 1436–1456. https://doi.org/10.1016/j.infsof.2014.02.012

Domingos, H. J. L., Martins, J. A. L., & Tecnologia, F. C. (1997). *Coordination and Tailorability Issues in the design of a Generic Large Scale Groupware Platform*.

Eaton, B., Elaluf-Calderwood, S., Sorensen, C., & Yoo, Y. (2015). Distributed tuning of boundary resources: The case of Apple's iOS service system. *MIS Quarterly: Management Information Systems*, *39*(1), 217–243.

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. https://doi.org/10.1111/isj.12186

Gawer, A., & Cusumano, M. A. (2014). Industry Platforms and Ecosystem Innovation. *Journal of Product Innovation Management*, *31*(3), 417–433. https://doi.org/10.1111/jpim.12105

Geertz, C. (1973). *"Thick Description: Toward an Interpretive Theory of Culture."* The Cultural Geography Reader; Routledge. https://doi.org/10.4324/9780203931950-11

Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: The boundary resources model. *Information Systems Journal*, *23*(2), 173–192. https://doi.org/10.1111/j.1365-2575.2012.00406.x

Gizaw, A. A., Bygstad, B., & Nielsen, P. (2017). Open generification. *Information Systems Journal*, *27*(5), 619–642. https://doi.org/10.1111/isj.12112

Health Information Systems Programme (HISP). (n.d.). Retrieved May 19, 2020, from https://www.mn.uio.no/ifi/english/research/networks/hisp/

Henfridsson, O., & Bygstad, B. (2013). The Generative Mechanisms of Digital Infrastructure Evolution. *MIS Quarterly*, *37*(3), 907–931. JSTOR.

HISP India. (n.d.-a). *About Us*. Retrieved April 2, 2020, from https://hispindia.org/Pages/About_Us/About_Us.html

HISP India. (n.d.-b). *Our work: By Region*. Retrieved May 22, 2020, from https://hispindia.org/Pages/By%20Region/By%20Region.html

HISP India. (n.d.-c). *Services*. Retrieved May 22, 2020, from https://hispindia.org/Pages/Services/Services.html

HISP Tanzania. (n.d.). *About us: HISP Tanzania*. Retrieved May 23, 2020, from https://hisptanzania.org/#/about-us

Klein, H. K., & Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, *23*(1), 67–93. JSTOR. https://doi.org/10.2307/249410

Li, M. (2019a). An Approach to Addressing the Usability and Local Relevance of Generic Enterprise Software. *Association for Information Systems*, *10*, 17.

Li, M. (2019b). *Making Usable Generic Software—The Platform Appliances Approach*. https://doi.org/10.13140/RG.2.2.11381.83687

Li, M., & Nielsen, P. (2019). *Making Usable Generic Software. A Matter of Global or Local Design?* 18.

MacLean, A., Carter, K., Lövstrand, L., & Moran, T. (1990). User-tailorable systems: Pressing the issues with buttons. *Proceedings of the SIGCHI Conference on Human*

Factors in Computing Systems Empowering People - CHI '90, 175–182. https://doi.org/10.1145/97243.97271

Martin, S. (n.d.). *Angular vs React vs Vue: Which is the Best Choice for 2019? | Hacker Noon*. Retrieved May 19, 2020, from https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847

McKenzie, H. (2013, July 23). *Move fast, break things: The sad story of Platform, Facebook's gigantic missed opportunity*. Pando. https://pando.com/2013/07/23/move-fast-break-things-the-sad-story-of-platform-facebooks-gigantic-missed-opportunity/

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. https://doi.org/10.1177/0306312706066022

Robbes, R., & Lungu, M. (2011). A study of ripple effects in software ecosystems (NIER track). *Proceedings of the 33rd International Conference on Software Engineering*, 904–907. https://doi.org/10.1145/1985793.1985940

Roland, L. K., Sanner, T. A., Sæbø, J. I., & Monteiro, E. (2017). P for Platform. Architectures of large-scale participatory design. *3-34*. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2478234

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. https://doi.org/10.1057/palgrave.ejis.3000700

Soh, C., & Sia, S. (2008). The Challenges of Implementing "Vanilla" Versions of Enterprise Systems. *MIS Quarterly Executive*, *4*(3). https://aisel.aisnet.org/misqe/vol4/iss3/6

Susman, G. I., & Evered, R. D. (1978). An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly*, *23*(4), 582–603. JSTOR. https://doi.org/10.2307/2392581

Tilson, D., Lyytinen, K., & Sørensen, C. (2010). Digital Infrastructures: The Missing IS Research Agenda. *Information Systems Research*, *21*(4), 748–759. https://doi.org/10.1287/isre.1100.0318

Tiwana, A. (2014). Platform Architecture. In *Platform Ecosystems* (pp. 73–116). Elsevier. https://doi.org/10.1016/B978-0-12-408066-9.00005-9

Villiers, R. (2012). *Models for interpretive information systems research, part 2: Design research, development research, design-science research, and design-based research - a meta-study and examples*. 238–255. https://doi.org/10.4018/978-1-4666-0179-6.ch012

von Hippel, E. (2005). Democratizing innovation: The evolving phenomenon of user innovation. *Journal Für Betriebswirtschaft*, *55*(1), 63–78. https://doi.org/10.1007/s11301-004-0002-8

von Hippel, E., & Katz, R. (2002). Shifting Innovation to Users via Toolkits. *Management Science*, *48*(7), 821–833. JSTOR.

Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, *15*(3), 320–330. https://doi.org/10.1057/palgrave.ejis.3000589

Wareham, J., Fox, P. B., & Cano Giner, J. L. (2014). Technology Ecosystem Governance. *Organization Science*, *25*(4), 1195–1215. https://doi.org/10.1287/orsc.2014.0895

Yazan, B. (2015). Three Approaches to Case Study Methods in Education: Yin, Merriam, and Stake. *The Qualitative Report; Fort Lauderdale*, *20*(2), 134–152.

Zahle, J. (2019). Data, epistemic values, and multiple methods in case study research. *Studies in History and Philosophy of Science Part A*, *78*, 32–39. https://doi.org/10.1016/j.shpsa.2018.11.005