UiO : Department of Informatics
University of Oslo

Software Requirement Management in Generic Open Source Projects

A Case Study of the Generic DHIS2 Software

Stian Sandvold

Master's Thesis Spring 2019

Software Requirement Management in Generic Open Source Projects

Stian Sandvold

1st February 2019

Abstract

Managing requirements are an integral part in any type of software development. In generic open source software projects, this process is increasingly difficult. Designing generic requirements and balancing the needs of different stakeholders are key to ensure the success of the software.

In this thesis, we look closer at the requirement management process in global generic open source software projects and the challenges they deal with. We present a rich case study of the DHIS2 software project and a conceptual framework to analyze and understand software requirement management in generic open source software projects based on the dimensions of methods of generification, models of innovation and methods of governance. Further, we discuss the case study using the conceptual framework, guided by the following two research questions: "How are requirements managed in global generic open source software projects?" and "What challenges exist in managing requirements in global open source software projects and how can they be dealt with?".

We finish the thesis by comparing the changes and challenges presented in the case study. Furthermore, we point out how the requirement management process adapt to the growth of the project and how it is impacted by the challenges.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, I will give a brief introduction to the motivation behind the thesis, my research questions and an overview of the structure of the thesis itself.

1.1 Motivation

Managing requirements is a critical process of software development which decides the shape and future of the software directly. Having a good process for managing requirements is essential for making the right decisions both for the software and for its users. Projects, where software is design based on a specific and clearly defined use-case, defining, handling and implementing requirements, might not be so challenging. However, when use-cases increase in number and complexity, the process involved in managing requirements becomes significantly more complicated. Open source software and generic software are examples of how a software's design can make working with requirements more complicated, and introduce new challenges in the process. Generic software, meaning it can be applied to different contexts, requires, for example, more effort when attempting to understand and define requirements depending on the number of existing use-cases. Presented in this thesis is a case study of the DHIS2 software, including its requirement management process and how that process has changed over time. The DHIS2 software is a global generic open source platform, which deals with a set of challenges not usually found in other software projects.

The thesis will provide insight into how one of the largest open source health information systems deals with challenges related to the requirement management process. By looking at the organization and the requirement management process both historically and contemporary, we gain an overview of how these have changed over the years. These changes, combined with our understanding of the continuously changing context of the implementation and development of the software, allows us to identify and understand the challenges that triggered the changes. The challenges and thoughts discussed in this thesis could potentially be applied to other similar projects to gain a better understanding of the

process of requirement management and how its context influences it.

Looking at how an established project like DHIS2 deals with requirements could be both insightful and valuable for those working in other projects. It deals both with challenges found in proprietary software projects, like generification, as well as open source projects, like balancing requirements from stakeholders provide funding and those who do not. Projects used for several different use-cases and across multiple domains also raises some interesting challenges related to the generification of requirements.

Through my role as a developer in the DHIS2 project, I have experienced first hand a lot of the challenges in requirement management. I believe this process plays a significant part in the success of the software. Without a good process, it would be hard to maintain a balance between requirements from different stakeholders and do long-term planning for the software.

My contributions from this thesis consist of two parts: A rich case study of the DHIS2 project and its requirement management process, and a framework to understand and discuss the requirement management process of software projects. The case study includes the history of both the DHIS1 and the DHIS2 software projects and a detailed description of the organization and processes related to the requirement management as it looks today. The framework I propose and use to discuss the requirement management process is a compilation of concepts presented in related literature. These concepts apply to different aspects of software development and can be used to describe both open source and proprietary software projects.

1.2 Research Question, Objective and Scope

My objective with the thesis is to try to gain a better understanding of the challenges related to managing requirements in global generic open source platforms. I attempt to describe how the process of managing requirements look in these projects and how to adapt to respond to the challenges mentioned above. In the thesis I will focus on the following two research questions:

1. How are requirements managed in global generic open source software projects?

2. What challenges exist in managing requirements in global open source software projects and how can they be dealt with?

To address these research questions, I will look at the DHIS2 software both historically and contemporary, describing the context of the development, the organization of the core team, the processes used to manage requirements and the challenges found here.

1.3 Overview of Thesis

Chapter 2 introduces some background information related to the DHIS2 software, which is the subject of this case study. It describes both the type of software, the organization overseeing the development and the properties of the software itself.

In chapter 3 I describe my role concerning the thesis, more specifically my role in the DHIS2 project. Next, I describe my research methodology and the process of choosing that and the topic of the thesis. Finally, I discuss how I gathered and analyzed data using in the thesis.

Chapter 4 presents the related literature I use in my thesis, how I intend to use it and what topics are relevant when describing the requirement management process later on. I summarize the chapter with a table listing the most interesting ideas I will apply later.

In chapter 5 I provide a brief history of the development of DHIS1 until 2006, and DHIS2 between 2004 and 2017, primarily focused on the requirement management process. After presenting the history, I provide a thorough description of the current state of DHIS2, including the ecosystem around it, the organization and the requirement management process itself. I summarize the chapter with a table outlining some key information about each of these periods.

Chapter 6 is where I combine the information about DHIS2 with the framework of ideas from related literature. For each period described in chapter 5, I describe the process of requirement management using the framework to provide a further understanding of the context, followed by the challenges found in those periods. I speculate as to the reason for the challenges, how they affected the development and how they were, or could, be solved.

In chapter 7 I conclude my discussions in chapter 6, and relate them to my research questions, followed by some thoughts about further research.

Chapter 2

Research Context and Background

This section gives a short description and introduction to DHIS2's software development process, including its use-case in global health, the HISP network and the software itself.

2.1 Monitoring and Evaluation in Global Health

Health management information systems (HMIS) are essential building blocks in health systems. Their primary purpose is to the collection and analysis of health data, and other relevant data, that can later be used to support health-related decisions in different processes. Without sufficient information, activities like planning, monitoring, and evaluation of efforts become difficult or even impossible. These systems can be used for a range of different purposes, for example, disease surveillance, stock management of medical supplies, as well as coverage and effectiveness of health-related efforts like vaccination or malaria-related spraying. HMIS, like DHIS2, is used for both these purposes and more, by the ministries of health in developing countries, organizations working with global health, to mention a few.

2.2 The Global HISP Network

The Health Information Systems Program (HISP) started as a pilot program in South Africa in 1994. The program included key-actors like the University of Western Cape, the University of Cape Town and the University of Oslo. In 2003, the original HISP team re-organized itself as a not-for-profit company, HISP South Africa. Since then, HISP has become a global network with a wide variety of actors, directly or indirectly supporting HMIS implementations and related projects around the world. The HISP group at the University of Oslo, HISP UiO, has since then played a central role in this network and hosts a team of software developers developing and maintaining the DHIS2 software. The HISP network

consists of various official and unofficial members, like NGOs, universities and HISP nodes like HISP Vietnam, Uganda, West Africa and more. HISP UiO describes the goal of HISP as:

> The overall goal of HISP is to enable and support countries to strengthen their health systems and their capacity to govern their Health Information Systems in a sustainable way to improve the management and delivery of health services. [3]

DHIS2 is a product of HISP efforts, but not all HISP activities are focused on the software itself. HISP organizations supports the strengthening of health information systems in countries in general, including capacity building and implementation support.

2.3 The DHIS2 Software

DHIS2 is an open source HMIS developed and maintained by HISP UiO. It is designed primarily for use-cases within the health domain, but due to the generic design of the platform, it has been adopted in new, different, domains like agriculture, education, and more.

DHIS was initially designed to replace paper-based systems, which has many challenges like the lack of flexibility, accessibility, and data-quality. The DHIS2 software is primarily used for gathering, analyzing and visualizing data, with support for both aggregate level data as well as individual level event and patient data. Common use-cases in the health domain include vaccination programs, maternal health programs, disease surveillance and more.

The DHIS2 software requires the users to configure so-called metadata. Information that describes data is what we consider metadata, for example, the data type, like text or integer, and what the data represent, like name or age. In the DHIS2 software, we use metadata to both understand, categorize and organize data. The process of designing and creating this metadata is referred to as customization and is an essential part of adapting the software to the local context.

We can simplify the types of content we have in DHIS2 to either being metadata that describes how we interpret data or data, meaning the raw data described by the metadata as mentioned earlier. Raw data is primarily organized using three basic types of metadata which answers the three following questions: What, where and when? The what and where is configured by the user in the form of the "Data Element" and "Organisation Unit" metadata respectively, while the when is automatically generated by the software as "Period."

Through the customization of the metadata, users can design forms for data entry using the metadata and design reports based on the dimensions of different metadata. The DHIS2 software comes with additional features that utilize the data and metadata to solve other use-cases than information gathering and reporting. Tools for improving data quality, assisting data

entry and data prediction is some of the other use-cases natively supported in the software.

In the DHIS2 software, there are two ways to work with data: Aggregate and Tracker. Aggregate data is, as the name implies, data reported aggregated — for example, a clinic reporting the number of deaths occurring in their facility for a given period. Tracker data, on the other hand, handles data on a more individual level. This data can either be one-time anonymous events, for example, when distributing bed nets, or tracked events, where data follows an entity over time, for example, a woman during her pregnancy. The Aggregate and Tracker data have some deviations regarding the metadata used to describe them, but overall work in a similar way as to how the software support the gathering, analysis of reporting of the data.

In addition to being a generic software, allowing multiple domains to utilize the same core features, the DHIS2 software is also considered a platform. As a platform it allows users to not only customize the metadata but also install custom applications to extend the core functionality. The DHIS2 software acts as a platform by offering an API and a core set of applications, allowing users to get started using the system quickly, or creating custom applications that can support particular use-cases.

With the generic and flexible design of DHIS2, customization of the software is required to adapt it to the local context. Customizing DHIS2 requires knowledge about both the local context, but also the software itself. HISP provides both training through academies and in-country implementation support to assist with the process of customizing the software. Through this capacity building around the software, the users also learn about how to use the data they collect, and how to change their organizational processes to utilize the data and make more informed decisions backed by real data.

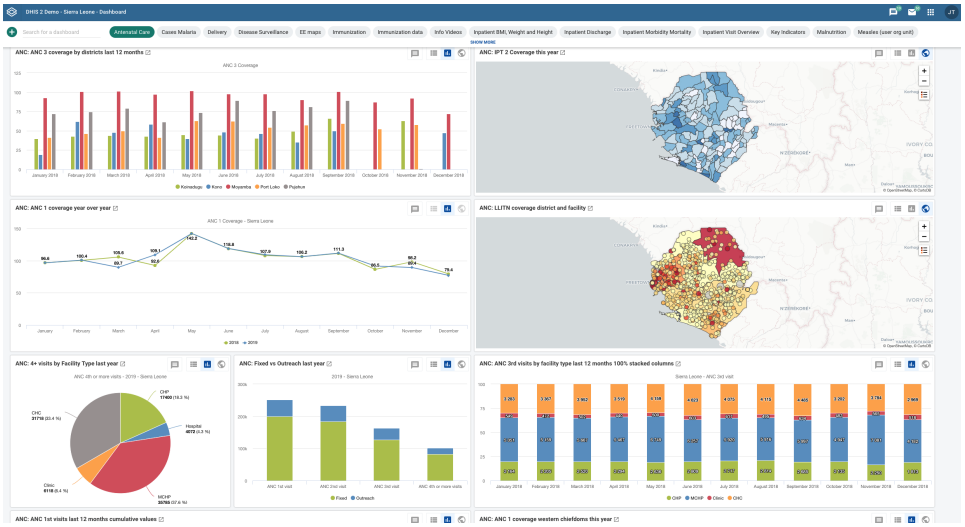

Figure 2.1: A screenshot of a dashboard in the DHIS2 software demo

Figure 2.1 is a screenshot taken from one of the instances of the

DHIS2 software, used for demonstration purposes by HISP UiO. The screenshot shows a Dashboard with different visual representations of data. The customization of the dashboard, the visualization and the metadata describing the data, are all part of the customization users go through to adapting the software to their local context.

DHIS2 was initially developed by HISP UiO in 2004, with the goal of modernizing the DHIS1 software. The DHIS1 software was an offline application, based on proprietary software like Microsoft Access. DHIS2 moved away from the proprietary software dependencies and was designed to work in an online environment, making it both completely free to use and much more accessible. Today, DHIS2 has become a global public good with over 60 countries and 23 organizations running implementations of the software, potentially covering over 1.3 billion people with their services.

The DHIS2 software requires third-party software to run, like the relational database system PostgreSQL and web-server supporting WAR applications, like Apache Tomcat. Both the third-party software and the DHIS2 software are available online and free to use. The DHIS2 software is available for download at https://www.dhis2.org/downloads or available for testing by anyone using the demo-instances hosted by HISP UiO found at https://play.dhis2.org/.

Because the DHIS2 software is available for anyone, free and globally relevant, it has been recognized as a Global Digital Public Good. A global public good, as described by Petter Nielsen & Sundeep Sahay [6], is a good that has "... a high degree of publicness, and are characterized by non-rivalry and non- exclusivity". Additionally, these are goods where "... the benefits are quasi universal across groups of people, social groups, places, and also generations".

2.4 HISP UiO

HISP UiO is the HISP organization located at the University of Oslo, Norway. The organization is responsible for the development and management of the DHIS2 software and consists of developers, coordinators, researchers, and students whom all contribute to the development of the DHIS2 software in some way. They oversee a range of different activities, including software development, capacity building, community management and collaboration projects with donors, all centered around the DHIS2 software.

In addition to the development of the core software, introduced in the next section, HISP UiO facilitates different capacity building activities. These activities includes both an online academy, a web-portal for learning about the software, as well as more advanced in-person academies, where the DHIS2 experts train and collaborate with users. HISP UiO also manages the DHIS2 community, which consists of mailing-lists, online forums, an issue-tracker and more. Donors are often interested in supporting the development of the software or funding specific projects to add support

for specific use-cases. HISP UiO manages both the funding of the DHIS2 core and the externally funded projects related to DHIS2 development.

2.5 DHIS2 Software Development

HISP UiO manages the development processes around the DHIS2 software. Requirements are both actively gathered by HISP UiO and submitted to HISP UiO by users from the community, who most often are people working in ministries of health, HISP nodes or NGOs. HISP UiO, or more specifically the team working with the DHIS2 software, review, accept and decide when to implement requirements in the software core.

Various donors support the development of the DHIS2 software through externally funded projects. Externally funded projects are temporary projects bound through a contract, describing the scope of work and deliverables. The projects are usually related to specific work, like supporting new use-cases, but can also include general development activities. Considering both the needs of donors and other users is essential to be able to make prioritization decisions that benefit both groups of users, so both donors and existing users are satisfied.

New versions of the DHIS2 software is currently released every six months. Cutting-edge versions of the software can also be downloaded directly from the source code repositories by users who want to try out the latest features right away. The period between each new version consists of minor segments referred to as milestones, usually lasting three or four weeks. Each milestone has a prioritized set of requirements planned for implementation.

After the release of a new software version, the core team shares a summary containing the latest changes with the community. Additionally, an updated version of the documentation of the software is released and included in the announcement. The last three versions of the software are supported by the core team, implying that the core team will only provide fixes for bugs found in these versions.

2.6 Summary

DHIS2 is an HMIS software developed and maintained by HISP UiO. The software is a global generic open source platform, which primarily is built around the health domain, but has traversed into several other domains. New versions of the software are available every six months, where the changes include a variety of requirements both from donors and free users of the software. Although the DHIS2 software is a significant part of HISP and the HISP network, the activities of the members of the HISP network is not solely related to DHIS2, but to strengthening the HIS in countries in general. Table 2.1 on the next page gives an overview of the topics discussed in this chapter.

HMIS/HIS	Health Management Information Systems are essential tools used to strengthen health systems. They gather, combine and analyze data from different sources to provide valuable information used in making health-related decisions.
HISP Network	The HISP network is a global network of universities, NGOs, HISP organizations and more. They focus on supporting countries to strengthening their health systems and building capacity in countries to govern their HIS sustainably.
DHIS2	DHIS2 is a generic open source HIS which works with both aggregate level data, like traditional HIS, but also individual level data. The software was primarily developed for the health domain but has several use-cases in other domains as well.
HISP UiO	The HISP UiO organization is based at the University of Oslo, Norway, and oversees the development of the DHIS2 software. HISP UiO has a range of different people like developers, implementation advisors, students, and researchers, whom all contribute to the development of DHIS2.
DHIS2 Development	All requirements for the DHIS2 software is reviewed, accepted, prioritized and implemented through the core team at HISP UiO. Organizations using or supporting the DHIS2 platform sometimes provide funding to the core development of the software or through externally funded projects.
Properties of the DHIS2 software	• Open Source • Generic • Platform • API & Core Apps • Global Digital Public Good • Funded through external projects

Table 2.1: Summary of the Research Context and Background Topics

Chapter 3

Methodology

In this chapter, I will describe my role as a DHIS2 developer and researcher, and how they relate, Additionally I will explain the research methodology I used and how I gathered and analyzed the data I present in my thesis.

3.1 My Role

During my master studies at the University of Oslo, I was offered a job as a developer, working part-time with the DHIS2 project. Soon after, I transitioned into a full-time position as a developer, and later as a team leader for the back-end developers. I have worked with the DHIS2 software since September 2015 and have seen a lot of the organizational changes, growth, and challenges along the way.

After a break in my master studies, I decided to write my master thesis about the development of DHIS2. My primary motivation for choosing to write about this subject was from my work. I saw the challenges the team is faced with when working with requirements for a project like DHIS2, which I found very interesting, and especially because I had been part of finding new ways to work with requirements when existing processes became insufficient.

The advantages of writing about this topic, concerning my role as a DHIS2 software developer, is the insight and perspective I can present. Additionally, I have both close access to the core team and knowledge about the organization and requirement management process. Through my role as a DHIS2 software developer, I get the opportunity to present and describe the process and challenges from a developers perspective, providing insight that might not be apparent looking at the project from the outside.

The disadvantages of my position, on the other hand, is my lack of experience with other projects than DHIS2. Except for working in a start-up for about two years before my master studies, I have no other involvement with professional software projects. This lack of experience could potentially narrow my perspective and make it more difficult to generalize my ideas to other projects.

Another concern I have about writing about a process I am a part of is

that my own opinion might influence my descriptions. During the writing of this thesis, I have worked with several members in the core DHIS2 team. Through the input they provided, I compare the different perspectives to find the most general description of the information. Some of these members of the core DHIS2 team also reviewed my final presentation of the information to confirm it was correct.

When I initially started working on this thesis, I only knew I wanted to take a look at how DHIS2 handle requirements. Without a more specific sense of what to focus on, my scope became too broad, so I tried to look at the various challenging and interesting aspect to address. Early in this process, I was considering the idea of writing about specific challenges of the requirement engineering process in DHIS2, like the specification of requirements, or how requirements are validated. I felt these topics quickly became too narrow in my own opinion and not quite right.

After I started finding some existing literature, and several meetings with my thesis advisor, I decided to move my focus away from the details of the requirements. Instead, I decided to look at the overall process requirements goes through and how they have changed over time and how different people influence them regarding prioritization and acceptance. With this scope, I can both describe the elements that initially motivated me to choose this topic, but also provide insight into the process that is the engine of DHIS2 development.

3.2 Research Methodology

The research method I used for this thesis is the case study methodology. More specifically, a soft case study as described by Kristin Braa & Richard Vidgen [1]. The overall goal by using this method is to give an understanding of the processes and challenges involved in requirement management in global generic open source platforms, such as DHIS2. I attempt to achieve this by interpreting both the history of the software and presenting a more detailed description of the current situation of the requirement management process.

A noticeable deviation from the characteristics of the soft case study method as described by Braa & Vidgen [1] is my participation, which is significant due to my role in the development of the software.

The circumstances motivated the reason for choosing the case study method for the thesis. After working with the project since 2015, the idea of taking a closer look at how both the organization and processes work and has adapted to new challenges was interesting.

I decided to base the thesis around my own experiences and observations, which initially lead me to consider an "action case" method for my thesis. Using the "action case" method, I would attempt to introduce a change in the process to deal with some of the challenges. This idea, however, was shortly abandoned because of the rapid ongoing changes made to the process and organization already. It would be difficult to both introduce a change, as well as to evaluate its impact, with several other changes

happening simultaneously in the organization and process.

Similarly, the idea to write the action case based on a change I had already been part of implementing, but trying to evaluate the impact would still be difficult due to all the other simultaneous changes. After shifting the perspective from introducing a change to understanding the motivation and reason for introducing changes, approaching the topic as a case study seemed more appropriate.

Through a case study, I would look at the development of DHIS1 and DHIS2 both historically and contemporary. By looking at the requirement management and the context surrounding the development, I could present an interpretation of the processes and challenges, and how the process and organization adapted to deal with these challenges.

By combining how both the context surrounding the development and the organization and process itself has changed over time using the case study method, I believe the discussions, conclusions and conceptual ideas I present can to some degree be generalized to apply in other similar projects.

3.3 Data Analysis

When I started working on the thesis, I searched for relevant literature to use. I used Google Scholar to initially look for literature specific to requirement management, using terms like "Requirement Management", "Requirement Engineering", "Requirements in Software Development". The literature I found focused very much on the details regarding requirements, including specific stages like the elicitation and validation requirements. Because I wanted to present the process of managing requirements, and the challenges it comes with, at a higher level, I felt this literature would not be very useful to me.

Through discussions with my thesis supervisor, we found several topics that work well with what I wanted to achieve with the thesis. "Open Source", "Platform", "Global software" are some of these topics. Based on these topics my supervisor also suggested relevant literature. When looking for literature on Google Scholar related to these topics, most of the results were related to the actual development of the source code, and less about the process and challenges related to managing requirements. Because the literature I already had covered several different aspects of the process I wanted to describe, I decided what I had was sufficient.

All of the literature is either describing a way of working with requirements directly, like generification or about processes that influence the requirement management process, like participatory design. Another important reason for the relevance of this literature is that they were touching on one or more topics that related to the DHIS2 software, such as "open source," "platform" and "global software" - Some even using the DHIS2 software as an example to describe their research.

After reading the literature, I chose relevant terms, methods, and processes and combined them into a framework. The objective of this framework was to describe the requirement management process

or aspects that influence it, using ideas already established in existing literature. Although the primary application for this framework is in the context of the DHIS2 software, it should be possible to apply it in other projects as well.

When describing the history of DHIS1 and DHIS2, the majority of information was collected from a paper by Jørn Braa & Sundeep Sahay, "Integrated Health Information Architecture" written in 2004. This paper describes both the history of DHIS1 and DHIS2. My description of the DHIS1 and DHIS2 history focused on the development and requirement management process, unlike the original paper which has a much broader approach. In addition to the information found in this paper, I filled in some gaps through informal interviews with members of the DHIS2 core team.

In section 5.2, there is an account of the requirement management process in DHIS2. This account is primarily based on my knowledge and experience from working as a developer, and later a team leader since 2015. All the information in that section comes from my understanding and observations, then supplied with additional information from, and later reviewed by, other members of the core DHIS2 team.

The requirement management process is continually evolving in DHIS2, and the members of the core team are adapting to the changes in the process. That means there are slight variations between the different teams as to how they approach the process in detail. To avoid describing multiple detailed processes, I disregard the minor deviations between the approaches and focus on the overall process to which all teams conform. Examples of details I have disregarded include how one team requires more detailed planning because other teams depend on them, or how product managers organize their requirements differently in the issue tracker. Regardless of these details, the overall process looks the same, and they provide no additional value to the description.

In table 3.1 on the facing page I provide an overview of my interactions with different members of the core DHIS2 team. These interactions relate to gathering information, either through informal discussions or interviews, or to validate my description of the organization, roles or the requirement management process. I did not track how long each interaction lasted but give an approximate estimate based on the number of times I interacted with them and the nature of the interaction.

Interactions marked as "Informal Interview" in table 3.1 on the next page refers to interactions with people where I had prepared a list of questions. These questions could be very open-ended on topics I had limited knowledge, like funding or the responsibilities of different roles, or they could be more specific when I already knew most of the information but was missing details. One informal interview was done in-person, and the answers were noted down, while the other informal interviews happened through online channels like instant messaging or email.

The informal discussions were sometimes planned, like when I needed more information about a particular subject, but was unsure exactly what. However, for the most part, they were unplanned, and usually the result of

Who	Lead Developer	Implementation Coordinator	Product Manager	Assistant Product Manager	Project Coordinator
Type of interaction	Informal Discussion	Informal Discussion, Informal Interview	Informal Interview	Validation	Informal Discussion, Informal Interview, Validation
Contributions	Information about the requirement management process, history of DHIS2.	History of DHIS1 and DHIS2, the DHIS2 eco-system, organization roles, projects, and funding.	The product manager role, responsibilities and how they work. Validation of the requirement management process and product manager role	Confirming my description of the requirement management process and the product manager role.	Information about projects and roles. Validation of the requirement management process, DHIS2 eco-system, and roles.
Where	UiO	UiO, Online	UiO	Online	UiO, Online
Estimated time spent	2-3 Hours over several interactions	3-4 Hours over several interactions	1 Hour	-	3-4 Hours over several interactions

Table 3.1: Overview of sources from the core DHIS2 team

other discussions, where the discussions were related to topics relevant to the thesis. The informal discussions were usually about specific stories, like developers traveling to work closely with the users or how we work with projects, where I asked follow-up questions about specific details. I did not take notes during the informal discussions, because they were mostly unplanned, but I confirmed the details of the information later when I included the information in the thesis.

All the information gathered through informal interviews and discussions was either confirming any assumptions I had made or was used to fill in gaps I had in my descriptions.

One way I tried to assure my descriptions where less influenced my own opinions, was by interacting with members of the core team that would have a different perspective. For example, by combining the information from me, a project coordinator and an implementation coordinator, I got a more general understanding of the information than I would if I only had one source. Another example is how my perspective of the product manager meetings bases itself on the "tracker" product, the meeting I usually attend, but work with a product manager from a different team to review and validate my description. In table 3.1 on the preceding page I summarize my various discussions and other interactions.

Chapter 4

Related Literature

Using the methods and mechanisms presented in this chapter, I would like to propose a framework that can be used to further describe the process of managing requirements, from after gathering them and until their implementation. It appears that the perception in the existing literature is that the requirements acquired through the different processes, for example, participatory design, usually results in requirements that can be generified and implemented without much work. In the case of the generic and open source platform of DHIS2, which has a highly diverse community of users with different needs, use-cases and resources, this is not always the case. Based on the existing literature introduced in this chapter, I want to suggest a framework for describing the process of managing requirements. Applying this proposed framework in the context of a software project should help outline various aspects of the development that influence the requirement management process.

4.1 Generic and Open Source Software

Pollock, Williams, and D'Adderio describe generic systems as "... [systems that] are used in diverse places and appear oblivious to the form, function, culture or even geography of organizations" [7]. These systems, or solutions "... [have the] ability to transcend their place of production that they are now described as 'generic' or even 'global' solutions." [7]. In other words, generic software is software that is not tied to a specific setting but can apply to a range of different settings. Particular solutions are the opposite of generic solutions and are tied to a specific setting, making it hard, or even impossible, to move it to other settings.

Open source software is software that is made freely available to everyone. That includes both access to the source code and the use of the software. Open source software projects are developed on a voluntary basis by developers who collaborate in internet-based communities. Unlike traditional proprietary software, the software is available for free, and there is no direct path to profit. That means the motivation for developing open source software is unlike that of proprietary software [4].

Today there are many examples of generic open source projects, like

DHIS2 which is the basis of this thesis, but also projects like Apache web server, Linux operating system or Wordpress content management system.

4.2 Platform Architecture

Roland, Sanner, and Sæbø points out that the platform software architecture is a prominent technical architecture that can be employed to tackle complexity. They describe platform architecture as "... [platform] where reusable and generic functions are bundled as a platform core while specific services are developed as compliments, called apps." [8]. Architecture usually refers to the layers and modules a software is comprised by and the interfaces between them. The authors also use the term architecture in their study to refer to the "structure of a socio-technical ecosystem that potentially both enables and constrains participation in design." [8].

In their study, Roland et al. introduces the terms design and use flexibility to describe the flexibility of the software for the different layers of the platform architecture. The design flexibility represents the flexibility of further development, while use flexibility represents the flexibility of the software to be used out-of-the-box for a range of different or unexpected purposes. They describe DHIS2, which is the subject of their study, as having three layers: The generic core, the bundled apps, and the custom apps. These layers are connected to the generic core layer using application programming interfaces (APIs) and software development kits (SDKs). The generic core, primarily developed and maintained by a core team of developers, has low design flexibility and high use flexibility. Bundled apps, also developed by the core team of developers, are designed similarly to the generic core with low design flexibility and high use flexibility. The custom apps are not developed by the core team, but by local developers, allowing them to tailor the platform to the local setting. This layer has high design flexibility, but low use flexibility.

The problem of scale in participatory design (PD), defined as "the number of distributions of heterogeneous settings, developers, users and uses of a (software) product over time." [8], is another issue they bring up in their study. They categorize different types of PD and how they relate to the scale, including singular, serial, parallel and community PD. Using the layer representation of the platform architecture employed in DHIS2, they show how each layer utilizes different types of PD and that they can coexist in the same project.

The singular PD is the most familiar type of PD, where developers and users work face-to-face to make design decisions, based on concrete work tasks.

In serial PD, developers and implementers work closely with end users and make on-site visits. The developers and implementers configure and customize the software to fit distributed, but relatively similar work practices in collaboration with end users. Implementers are often filling the role as mediators between users and developers to translate requirements to a more technical description.

Parallel PD is where developers, implementers, and customizers make short field visits and arrange workshops with user representatives to interact with users. Power users and local experts play a central role in gathering requirements, and multiple projects undergo parallel design processes.

The final classification, community PD, use power users and domain experts to represent end users in meetings and workshops. The focus is on the appropriation of the software access, diverse uses, and settings, and the core developers make all final generic product decisions.

4.3 Generification

The term 'generification' is described by Pollock et al. as "The practice of making software generic (generification work), including its various explicit and revealed generification strategies" [7]. They bring up a range of challenges related to the process of generification like particularization, 'design from nowhere', and complying to the diverse needs of the community. In addition to these challenges, the software has to remain attractive for both existing and new users. Through their study, they follow two software packages and identify three methods for generification utilized by the suppliers of these software packages: Management by community, management by content and management by social authority.

Pollock et al. [7] explain the management of community as methods for utilizing a community of users to discover, discuss and get feedback on needs. The idea is to bring the design process from the private locations of the users to a public forum, where users can compare their needs and decide on a set of similarities that can be generified. Another positive effect of this shift in context is that users' attitude towards the overall generification process improves and they gain an understanding of the challenges regarding generification.

Management of content is the second category Pollock et al. [7] describe. While the suppliers want to add features that support as many users as possible, complying to users' every single need will cause the software package to become particularized, making the software package unusable in other settings. Therefore there is a need to shape and smooth the diverse needs of the users and sift out requirements that are too specific, to maintain a generic software package. The suppliers apply different methods for managing the users' needs, like process alignment, where users can align their processes to fit specific templates, recognizing generic features or organizationally particular needs, allowing the supplier to enforce boundaries of the software package. These methods shift the burden of generification to the users, requiring them to work together to discover generic solutions to their needs without relying on the supplier.

The final method Pollock et al. [7] describes, is the management of social authority. When the community grows too big for the supplier to interact with publicly, and interaction with individual users too demanding, the supplier makes a change in how they interact with users.

In the study, the supplier categorized users into one of three categories: Transactional, consultative and strategic customers.

Transactional customers are those who want everything for free, and therefore the resources invested by the supplier into their needs would not yield any additional value. The supplier actively distanced these users from contributing to discussions about new features or the future of the software package.

The other categories of users, on the other hand, was frequently included in these discussions and quizzed about their opinions. These users would bring additional value to the supplier, like the consultative customers who want to work and spend with the supplier or the strategic customers who share the future vision of the software package and where it is going in the coming years.

In addition to classifying their customers according to the value they contribute, the supplier also placed customers closer or further away relative to themselves according to their willingness to do organizationally changes to align with the software package and who they considered "good generifiers". Their evaluation of "good generifiers" was based on when they adopted the software package - early-adopters being placed closer, while late adopters further away. The closer they positioned users to the supplier, the more they were included in the design and feedback process of the development and the overall future of the software package.

Building on the work of Pollock et al. [7], Gizaw, Bygstad and Nielsen [2] acknowledge the need and feasibility of generic software, but propose an alternative approach to generification they refer to as 'Open generification'. Their concern is that the methods described by Pollock et al. characterize the generification process as "a closed process, controlled by the vendor with little, if any, room for innovation as a distributed activity" [7]. Walsham referred to in Gizaw et al. [2] points out potential issues with different approaches to generification where "The dominant vendor controls the policies of generification, the software architecture and the process of final product development" [2] like which users can influence the generification process and design, making the system more suitable for some than others, and whether the consequences of this are positive or negative.

Gizaw et al. [7] address the concerns raised by Walsham by referring to the governance mechanisms top-down-centralized cathedral model and bottom-up-distributed bazaar model previously described by Raymond and Capiluppi & Michlmayr. The Cathedral method is a top-down-centralized approach, usually employed when working with proprietary software, where participation is restricted to an exclusive group. On the opposite side, the Bazaar method is a bottom-up-distributed approach in which "The public is allowed to participate and gain authority and leadership through their contributions and knowledge of the software under meritocratic norms." [2], and where the development is conducted online as a transparent public process.

Open generification as described by Gizaw et al. [2] takes a different stance when it comes to managing requirement than what Pollock et al.

[7] present through their three core mechanisms. Unlike management by community, where the general idea is to shield the generic software by moving the design away from the local users, the open generification method leverages the local developers to modify the software package to fit their local needs. This approach allows for local innovations that global developers can include into the core software through a process of embedding and disembedding. The authors describe the process of disembedding as "incoming user requirements are translated into common requirements through abstraction and negotiation" [2]. Embedding, on the other hand, refers to the process of taking a global software package and adapting it to the local setting. This happens in one of two ways: By filling or configuring missing parts, intentionally left open by the global developers, or by local innovation to make the software work for the local needs not anticipated by the global developers. By leveraging local innovations and allowing global developers to include these innovations in the core software through a process of embedding and disembedding, more users can benefit from the innovation of other users.

Another interesting idea about generification Pollock et al. points out, is what they refer to as the 'generic examples of the particular'. The authors use this term to describe "particular features that aid the circulation of the package" [7]. This implies that even generic software packages can benefit from introducing particular features that could help both existing settings and potentially new settings.

4.4 Requirement Management in Open Source Software

In their study, von Hippel & von Krogh presents two prevalent models for innovation organizational science: The "private investment" and the "collective action" models. They also introduce the phenomenon of open source software project development to point out the lack of a third model that could more fittingly be used to describe these projects. They propose a new model, a compound between the "private investment" and "collective action" models, called "private-collective". [4]

The "private investment" model for innovation, as described by von Hippel & von Krogh, assumes the innovator is motivated by private goods and regimes of intellectual property protection and that innovators will avoid 'spillover', regarded as a loss of profit, to ensure the returns are appropriated from the investment. This model is common for proprietary software where development is limited by a budget, and the innovation is owned by the innovator and not freely revealed to society. [4]

On the other hand, the "collective action" model for innovation assumes innovators collaborate to produce public good during market failure. It requires that contributors give up control of knowledge they have developed for a project and make it a public good by unconditionally supplying it to a common pool. Unlike the "private investment" model where the motivation for innovation comes from the investment, this model en-

counters challenges regarding the motivation of potential contributors. Another issue with public goods is "free riders", who waits for others to contribute and reap the benefits without contributing themselves.

The new compound model proposed by von Hippel & von Krogh is the "private-collective" model for innovation. The model is a combination of the two former models and aims to cover some of the gaps left by the former models concerning open source software development. Participants in this model use their resources to privately invest in the projects and choose to reveal innovations freely as opposed to claiming proprietary rights. [4]

Another point von Hippel & von Krogh brings up is that open source software development stands out from the more traditional propriety software development. The most significant differences being that the contributors and innovators are more often users than just manufacturers and there is no direct path to profit as the product is free. The motivation for participation, on the other hand, can be indirect profit by increasing the sales of a related product, gaining experience or acknowledgment and social status. [4]

4.5 Summary

The process of generification will in some cases be performed by, if given the opportunity, the community as described by Pollock et. al., or through different types of interactions between a supplier or a core team and the users, for example using methods like a variety of participatory design types as proposed by Roland et. al.. However, as the scale of the software core grows, and the diversity in use-cases increase, generic requirements might not be generic enough to be accepted into the software core. For several reasons, the existing features and code of the software core should be reused, for example, to avoid growing the source code too big to maintain or to avoid adding features that are too particularized to be used by most users. For that reason, requirements which would generally be considered generic could be rejected for not being generic enough, or be subject to further generification to align it more with the existing software core or other existing requirements. This process can be described using the methods of generification as presented by Pollock et. al. and the extension of this, open generification, described by Gizaw et. al., and both influence the process of prioritization.

Prioritization of requirements is the process of deciding when, or if, a requirement will be implemented into the software core. There are several variables to consider in open source software for this process, including community, social authority and content as Pollock et. al. presents and Gizaw et al. further extends on, as well as the model of innovation described by von Hippel & von Krogh. An essential challenge when managing requirements in large scale generic open source software, is to make sure that the software is relevant for both existing and new users, while at the same time be able to acquire means to maintain and further

develop the software core.

The final process of implementing a requirements goes beyond the practical process of writing code and committing it to the software core, but also includes making sure that the users can understand and use it. A common challenge when implementing generic requirements that can support a range of diverse use-cases is that it requires customization by the users to fit their local needs. This need for customization means that in addition to just adding the feature to the software core, the core team needs to communicate with and educate the community about the new feature before it can be used. This activity is often accomplished through the use of documentation, newsletters, mailing lists, workshops, and training as well as on-site implementation and configuration support.

Table 4.1 on the next page summarize the framework I will use to describe the requirement management process of the DHIS2 software. Each method, model and process can be used to describe something that directly, or indirectly, influence the requirement management process, which in turn can be used to describe how the process works or how it deviates from the descriptions in existing literature.

Methods of Generification	
Management by community	How the supplier utilizes the community in the generification process.
Management by content	Leverage the content of the software to discard or generify requirements.
Management by social authority	Segmenting the users to identify and leverage strategically valuable users.
Open generification	Focus on local innovation and global implementation through the processes of embedding and disembedding.
Models of Innovation	
Private investment	The intellectual property and returns from the innovation is the goal of the investment.
Collective-action	Contributors unconditionally give away their knowledge to a common pool to create a public good.
Private-collective	A mix of private investment and collective-action. Participants invest their resources in the project and freely reveal innovations.
Methods of Governance	
Cathedral	A top-down, centralized approach to governance with participation restricted to a small group.
Bazaar	A bottom-up distributed method of governance where the public participation is allowed and authority is gained through participation under meritocratic norms.

Table 4.1: Methods, models and processes included in the suggested framework used to describe the requirement management process in software development.

Chapter 5

Results

In this section, I will start by describing how requirement management looks throughout the history of DHIS1 and DHIS2. In these descriptions, I will both to point out how the process has adapted to a changing context and new challenges, but also to show the fundamental ideas that have been a part of the process since the start. This history of DHIS1 and DHIS2 is primarily a summary, heavily based on the work of Jørn Braa & Sundeep Sahay [9], then further expanded through informal interviews with members of the DHIS2 core team. Finally, I will go more into detail about the flow of requirement management and the different people and roles that make up the requirement management process as it looks today.

5.1 History of DHIS1 Requirement Management

DHIS1 was a digitalization of a paper-based system and development started in South Africa in 1994. The team developing the software consisted of 6-10 people, in which only two were programmers. The software was open source and free, and written with C#, but used Visual Basic and Access, requiring the users to have the MS Windows and MS Office stack to be able to run it. HISP, at the time known as HISPP (Health Information Pilot Program), was overseeing the development and was piloting DHIS1 in 3 districts in South Africa. Working closely with the district health management team (DHMT) with users like health workers, health information officers and managers, the developers developed the software with a combination of rapid prototyping and participatory design. This combination resulted in rapid releases of new versions of the software package, sometimes as frequently as every day, and played a significant role in the success of the software.

The process of reporting feedback and new requirements to the team would either happen directly with developers or through other staff involved with the development. Any user could provide feedback or requests to the development team, and access to the team was given in a meritocracy-like fashion. That means a user's access to the team and their influence in design decisions was based on being innovative or on their interest and involvement. Although the requirement management

seemed to work well in this period, a lot of the processes was based on improvisation, and there was no specific structure.

The developers and users would sit together and discuss new requirements, and developers would often have to guide users to help them understand what their needs were. Taking into consideration the number of users requesting a new feature, the capacity of the developers and the overall benefit of the feature, the developers prioritized which requirement they would implement. Although they didn't choose to implement all requirements, all of them were recorded, regardless if they were implemented or not. By the end of the period of initial DHIS1 development, the frequency of the rapid releases of new versions would slow down to focus more on stability and less on new features.

The success of the software could be said to be the result of the close collaboration the team had with the users. With the combination of rapid prototyping, participatory design and quickly responding to users feedback, users became encouraged and more involved in giving feedback to the team. Also contributing to the success, was the decentralization and local empowerment goals of the post-apartheid reform in South Africa at the time. This reform influenced the design of the software, resulting in a flexible and robust solution that worked well in the context of South Africa where the health system and data requirements were extremely fragmented and continually changing.

After three years of intensive development, the first three pilot districts would eventually result in a national implementation in South Africa in 2000. Seeing the success of the software in South Africa, new countries like Cuba, India, and Mozambique wanted to start pilot projects for DHIS1. These new pilots introduced new challenges for the developers and implementers, primarily related to political barriers in Cuba and India, but also for more technical reasons like in Mozambique which changed the development of DHIS1.

While DHIS1 was successful in South Africa, being generic enough to travel across districts and scale up to a national level, this proved problematic in Mozambique. The DHIS1 software was too heavily tailored to the South African implementations, making it very difficult to implement in Mozambique. A significant difference in this pilot from the South African pilots was that the developers, still located in South Africa, were now unable to interact with the local users in the same face to face participatory design fashion as they had been able to in South Africa, making it more challenging to provide the same level of support.

Although more challenging than before, by 2005 DHIS1.4 was released. During this period, the developer team was still located in South Africa and would travel to the local implementations to discuss requirements before traveling back and working on the software.

5.2 History of DHIS2 Requirement Management

In 2004 the first lines of code were written for what would become DHIS2. The University of Oslo would coordinate the development of DHIS2 and focus on modernizing the DHIS1 software. The modernization of DHIS1 would include using bleeding edge technologies like Java and turning it into a web-based platform. Unlike DHIS1 which required proprietary software like MS Windows and MS Office, DHIS2 was not dependent on any proprietary software to run.

The developers who worked on the existing DHIS1 software did not work on the DHIS2 software, as they were busy working on the DHIS1.4 version. It was one of these original developers of DHIS1 who suggested that HISP UiO would oversee the DHIS2 development. Between 2004 and 2011, contributions to the DHIS2 software were mostly made by master students, as well as some research staff and teachers, at the University of Oslo.

Several new tools were adopted during this period to support working in a distributed environment, including Bazaar for source code management, Launchpad for issue tracking and mailing-lists for communication and support. Unlike DHIS1, DHIS2 would now be available through a centralized code repository, enabling anyone to retrieve and submit code changes online. This idea of supporting a distributed working environment was one of the ideas behind the development of DHIS2. The idea was to have a core team at HISP UiO, and distributed teams located closer to the implementations to work on the adaptation and customization of the system to fit the local needs.

The significant change in technology and tools did cause some unexpected challenges. Developers who had been formerly working on DHIS1 and used to working with C# had issues with moving to a Java environment, and some were unable to make the transition. Another issue was the introduction of an API-based design. Because DHIS1 experts worked with a database-driven software, like DHIS1, they were confused by how to find information using the API. In addition to HISP UiO, a team of developers from Vietnam who had experience working with and customizing the DHIS1 software was doing the majority of customization work for DHIS2 implementations.

Until 2011 there were very few countries using the DHIS2 software. The most notable implementations at this point include India and Sierra Leone. Around 2011 HISP UiO received external funding to support the development of the DHIS2 software, which allowed the hiring of 4 full-time project staff. With four full-time developers working on the project, 2011 marks the beginning of the core team, a non-research externally funded development team at UiO.

Between 2011 and 2013 one of the developers of the core team was traveling back and forth between Kenya to support a national scale online implementation of DHIS2. The developer worked with HMIS staff in the ministry of health in Nairobi, designing and fine-tuning features they needed, traveling to Kenya every 2-3 months over these two years. After

Kenya successfully scaled up in this period, several other countries like Uganda, Ghana, and Rwanda followed, and DHIS2 began attracting the attention of international donors.

The most significant growth of the DHIS2 software happened after 2013. PEPFAR (President's Emergency Plan For AIDS Relief, https://www.pepfar.gov/) choose DHIS2 as their internal reporting system and provided funding to the core DHIS2 team, allowing the team to grow. GF (Global Fund, https://www.theglobalfund.org/en/) received a lot of requests from countries for DHIS2 support and also provided funding to the core DHIS2 team to facilitate this, including both software development, but also an implementation team. Subsequently, more and more donors and countries adopted DHIS2 after this.

The strategy behind the development of DHIS2 was to have it open and transparent. All issues were listed on the online issue-tracker, Launchpad, where anyone had access to view, submit and discuss on issues. New requirements would usually be submitted to the issue-tracker by advanced users who have a good understanding of the system and domain, HISP coordinators who are supporting the implementations, the developers themselves, or as the result of discussions found in the mailing-lists. When a new feature was recorded, the lead developer would review and prioritize the issue.

In the following years, the small core team of DHIS2 developers grew from a handful of people, primarily located at the University of Oslo, to having over 30 developers both working locally and remotely. To keep up with the growing number of users and interest for DHIS2, staff was hired to coordinating projects and manage requirements. Additionally, new tools like Jira for issue-tracking, GitHub for hosting the source code and Slack for instant messaging were introduced to improve the collaboration of the growing team.

During this period of rapid growth, the organization had to change the current routines and processes to adapt to the new situation. The first issue was the core team's size. Initially, the lead developer was the person primarily responsible for assigning work to developers, supporting them and following up on the work. That implies the lead developer would usually have the final say in prioritizing requirements. The delegation and prioritization of work were done either by face to face interaction with individual developers or as part of the weekly developer meeting. These weekly developer meetings included all the developers, and any current work, or issues developers had, would be discussed and resolved during the meetings. Because the developers had different parts of the software they worked on, and the number of participants in these meetings kept growing, these meetings quickly became unproductive for most developers. With more developers joining and more requirements raised by the growing number of users, the core team of developers split into different technical teams. These teams grouped developers by the nature of their work on the software; each team supervised by a team leader. Two of these teams, the Backend and Analytics teams remains the same to this day, while the Android team moved out of the University of Oslo

and the Frontend team would later change into the Apps team. The teams would arrange team meetings, similar to the previous developer meeting. This new team structure also means some of the burdens of assigning and following up on work was relieved from the lead developers and the responsibility shared between the lead developer and the team leaders of each team.

A second challenge, tied to the first, is the fact that it became increasingly difficult to review requirements from all the different stakeholders and to discuss and prioritize these requirements for the lead developer alone. As a result, the most critical stakeholders would receive priority, because their requirements are often related to funding or critical for large implementations. The implications of this were that it could become difficult for stakeholders with limited opportunities to offer funding to have their needs heard among all the different requests the core team received. People with good knowledge of the software and health domain was given the role of product managers. Product managers would manage the review of requirements and prioritization of the different part of the software, trying to organize and get an overview of the growing number of requests. Another initiative was also launched to improve the communication with ministries of health, named RCAT (Roadmap Country Advisory Team), which works indirectly with ministries of health, through HISP nodes, in countries to make it easier to understand their needs and prioritize them.

5.3 Requirement Management in DHIS2

By 2018 both the DHIS2 software and the surrounding ecosystem has evolved and grown to a different scale than what DHIS1 or the initial version of DHIS2 was. With a growing number of implementations of all sizes, use-cases, and domains, trying to manage requirements becomes a considerable challenge. The core team overseeing the development of DHIS2 has grown from being a small team to having over 50-60 people, including developers, researchers, coordinators, and others, working both locally and remotely. In this section, I will give insight into the organization and processes related to the development of DHIS2, as well as the surrounding eco-system like the DHIS2 community and community platform.

5.3.1 The DHIS2 Innovation Eco-System

In this section, I will introduce and describe what I consider to be the eco-system surrounding the DHIS2 software. By eco-system I more specifically refer to the stakeholders, tools and services, activities and processes surrounding the software, both directly or indirectly contributing to the overall development of the software. Jørn Braa & Sundeep Sahay presents their interpretation of the DHIS2 innovation ecosystem with figure 5.1 on the following page, which I will further expand on in the section to also include tools and services which help facilitate the activities

of the ecosystem, but also how everything relates to the requirement management process.

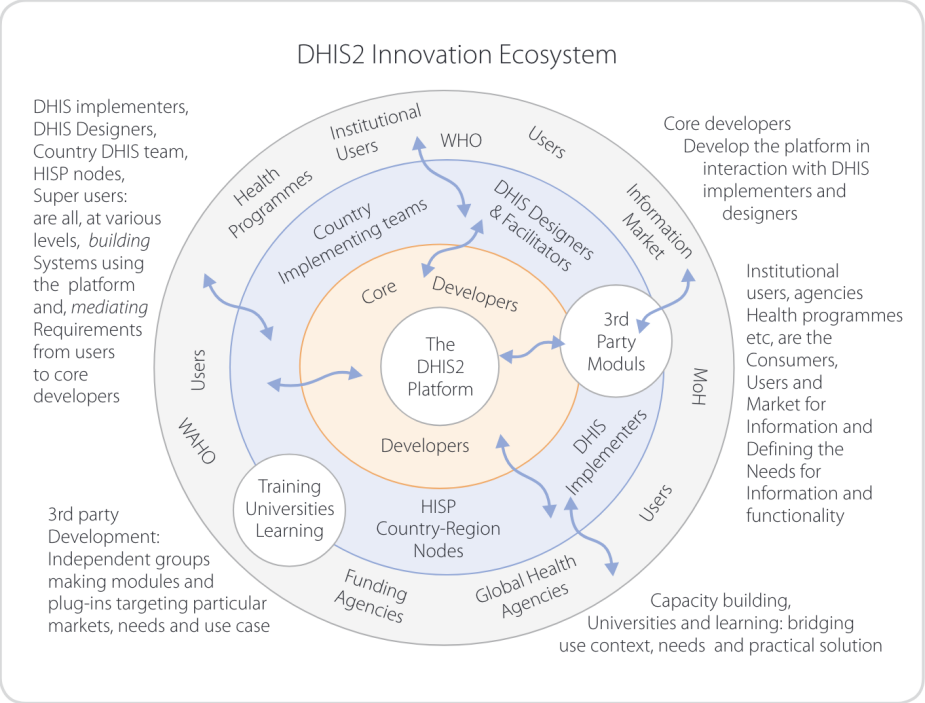

Figure 5.1: Figure taken from "Integrated Health Information Architecture" by Jørn Braa & Sundeep Sahay, illustrating the DHIS2 innovation ecosystem. [9]

Core Developers

As illustrated in figure 5.1, the core developers are the actors closest to the DHIS2 software. The different roles, responsibilities, and processes they follow can be found in section 5.3.2 on page 36, but from the perspective of the ecosystem, all interaction with the DHIS2 software happens through the core-team. The core team, which also include DHIS2 coordinators working at the University of Oslo, are also considered to be part of the DHIS2 community, alongside with other stakeholders working with DHIS2.

The DHIS2 Community

The community surrounding DHIS2 includes anyone working with DHIS2, including core developers, ministries of health, third-party developers, donor organizations and more. In addition to these different actors, some of which will be described in more detail further down, the community would not be the same without the community platform HISP UiO provides to enable interaction between its members. This community platform, facilitated by tools and activities primarily organized and managed by

HISP UiO, is used by the community to ask for support, present their needs, share experiences and learn from each other.

The core tools and services managed by the DHIS2 core team, includes mailing-lists, source code repositories, a forum, and an issue tracker. The Discourse platform, now acting as both a forum platform and mailing-list, is the primary tool used to communicate publicly with the community. Several mailing-lists existed until they moved from Launchpad to Discourse, including one aimed at the users and community in general, one aimed more toward developers and more technical discussions and one revolving around a specific use-case - Disease surveillance. Now, the mailing-lists are part of the Discourse platform, which categorizes posts or emails into different categories and sub-categories.

Discourse, which replaced the mailing-lists, primarily works as a forum with a set of features making it very similar to stack-overflow, a popular website for developers to share experiences and ask for help. The structure of the DHIS2 Discourse forum makes it easier to find existing discussions. Discussions are grouped by categories and sub-categories with specific topics and are maintained by a team of moderators under the supervision of HISP UiO. Although Discourse is primarily a forum application, it also supports interaction similar to traditional mailing-lists, so the transition from mailing-lists to Discourse has little impact for users who decide to keep using it as a mailing list. The mailing-lists, and now Discourse, are primarily used to seek support from other users or the DHIS2 core team, both technical question and implementation-related question, post announcements from the DHIS2 core team, and discuss use-case specific challenges in one of the sub-categories, for example, disease surveillance. By participating in these discussions, the core DHIS2 team are able to support the community, but also gather requirements and find gaps in the software that they can report in the issue-tracker and implement later. For example, one user asked how to achieve a specific configuration in DHIS2, but it was not possible to achieve this configuration with the current version of the software. This discussion can then be picked up and discussed by the core DHIS2 team to evaluate the possibility to add this as a new requirement and implement it in a new version of the software.

Jira is the issue-tracker used and managed by the DHIS2 core team. It is where users can go to view, create and discuss features, requirements and bug reports. Users can also follow the progress of issues, and see an overview of planned changes to the software. For example, users can find issues like "Deduplication service" with a description of the issue, related issues and the current progress of the issue. Unlike Discourse and the mailing-lists, Jira requires a more technical ability to use proficiently and typically used by expert users, third-party developers or donors in addition to the core team itself. For that reason, most of the simple questions around features and bugs are first discussed in other channels like Discourse. Jira is used on a daily basis by the core team, and the information there reflects their work, like which issues are being worked on, which are planned, and the status of each issue. In addition to developers updating the status of

the issues they work on, product managers and team leads make an effort to review and update issues as well.

The final tool that helps facilitate the community is the source code repositories, hosted using GitHub. Source code repositories are repositories where the source code of projects is stored, providing version control of the code. All of the DHIS2 source code, including both the core server software and the core apps that make up the foundation of the platform, are hosted on GitHub. Through GitHub, the community can at any time download the source code and run it on their machines. Using the features of GitHub, the source code is branched out to indicate a new version of the software and tagged to mark minor version changes later on. Branching out means making a copy of the code which will have to be maintained separately from the master branch, the branch where all new code is committed to be included in the software. For the most part, only the core developers interact with these repositories, but it is also used to review and accept contributions to the software core from external developers. As the core team manages the repositories, external contributors are unable to change the source code directly but submit requests to the core team using GitHub. These requests will be reviewed by the core team, to make sure they align with the guidelines and fits into the software in general.

DHIS2 Academies

HISP UiO, along with other HISP nodes, organizes events called DHIS2 academies. These academies, excluding the web-portal for learning the DHIS2 basics called "Online Academy", are hosted in different countries, primarily in Africa and Asia. They are an opportunity for users to learn about specific DHIS2 topics, meet other members of the community, share their experiences or interact with DHIS2 experts. For the core team, these academies are the most valuable way to learn about the community, the different use-cases, and understanding user needs. When participating in academies, the core team can work face-to-face with the users and see their problems first-hand. In addition to the online academy, which is sometimes referred to as "level 0", and maintained by HISP UiO, there are three levels academies.

Level 1 academies are primarily focused on essential topics related to the use of the software, like "Design & Customization", "Information Use" and "Tracker", and are hosted by HISP partners. There are several level 1 academies hosted every year. These academies usually don't have members of the core team participating, but are hosted by expert users from the HISP partners, who can answer any questions related to the material presented at this level. The training material for level 1 academies is rarely affected by new features implemented in the software, as they usually only cover the core features which seldom changes.

The next level of academies, level 2, usually require certificates for completing relevant level 1 academies. The University of Oslo always hosts these academies and often focus on specific topics of DHIS2. These topics include "Server administration", "App development", "Implementation

strategies" and "Disease Surveillance". Members of the core team are often sent to these academies to either present features, learn about the use-cases, understand and solve user difficulties, or networking with users. During these academies, new features relevant to the topic are presented to the participants. These are often presented in combination with a Q&A session, to answer any questions the participants have about these new features. Depending on the relevancy of the feature to the topic of the academy, learning how to use the new feature could be added to the training material. For example, the level 2 academies covering the topic of the disease surveillance use-case introduced material for new features that directly solved requirements for this use-case after including it in the software. During these academies, there is usually plenary sessions where participants present their experiences with the software, which helps the core team improve their understanding of the use-case and come up with new requirements to extend the software and further support the use-cases.

The expert academy is the largest academy, having over 170 participants attending in 2018, and is hosted annually by the University of Oslo, at the University of Oslo. During the expert academy, most of the core team participates, both by hosting different sessions, attending sessions and networking with the participants. In addition to the core team, expert users, representatives from ministries of health, donor organizations and other significant stakeholders also participate in the expert academy. Unlike the other academies, the sessions hosted during this academy are hosted by both the core team and participants themselves. The structure of these sessions usually allows for an initial introduction to a topic, then opens up for a plenary discussion on the topic. The topics discussed at this event is usually at such an advanced level, that primarily people with an in-depth knowledge of both DHIS2 and the topic benefit from participating in the discussions. The work done in each session helps shape the future of the software, from solving existing problems to expanding into new use-cases and domains, which means the participants can influence the requirement management process to a certain degree.

HISP Nodes

HISP UiO is overseeing the development of DHIS2 and related activities, but it is not the only HISP organization working with DHIS2. Other HISP organizations, usually referred to as HISP nodes or HISP partners, are organizations located in different countries working to strengthen the health information systems in countries. A Memorandum of Understanding containing 19 principles represents the guidelines HISP nodes follow as officially recognized HISP nodes by HISP UiO. These principles relate to strengthening health information systems, developing free and open source software, acknowledging all outputs emerging from project and research, and more. Table 5.1 on the next page list all the official HISP nodes. These HISP nodes is an excellent resource for the core team, as they help organize local DHIS2 level 1 academies, support ministries of health with the DHIS2 implementations, work with NGOs and mediate requirements

HISP Nodes
HISP-VN (Vietnam)
HISP Bangladesh Foundation
Health Information Systems Program - Uganda
HISP India - Society for Health Information Systems Programmes
HISP Tanzania
HISP Nigeria
HISP West and Central Africa
HISP (Sri Lanka)
HISP Rwanda

Table 5.1: List of official HISP nodes [5]

between the core team and the users.

The members in these HISP nodes usually include a variety of different expertise, both in the DHIS2 software, health domain, software development, and server administration. Among other health information strengthening activities, they use their expertise to assist local implementations of DHIS2. The support they provide includes activities like server administration, customization of DHIS2, and development of custom DHIS2 apps.

Due to their role working with both ministries of health, NGOs and the DHIS2 core team, the HISP nodes usually gain a unique understanding of both the use-cases and the technical aspects of the DHIS2 software. That makes them invaluable as a source of information about requirements for the core team. Members from these organizations frequently submit new requirements to the core team and are often included in the process of designing said requirements. For example, members of both HISP South Africa and HISP Uganda is actively working together with the core DHIS2 team to gather, define and improve requirements related to the disease surveillance use-case of the DHIS2 software.

Expert Users

There are no official titles, roles or lists that distinguishes an expert user from other users. Expert users refers to members of the community who has asserted their knowledge of the DHIS2 software by supporting the community and contributing to the software. These users, sometimes referred to as power users in other literature, are supporting both other users and the core DHIS2 team through their contributions. The contributions mentioned includes supporting DHIS2 implementations, answering questions and supporting other members of the community, specifying requirements for the core DHIS2 team and more.

Most users the core DHIS2 team consider expert users today, are users working in the HISP nodes. These users are naturally in a unique position to gain knowledge about both use-cases and the software, and through their work with academies become highly qualified to answer

most questions from the community. However, the core DHIS2 team recognize other users as experts as well, regardless of their association with HISP nodes. Any user who provides significant contributions to the software or the community can be considered expert users.

The authority expert users gain through their contributions are not measured in any formal way. However, the new DHIS2 community forum, Discourse, enables users to vote on posts, allowing active contributors to gain recognition for the efforts from other community members. In the eyes of the core DHIS2 team, expert users have varying degrees of authority depending on the context. This authority usually refers to their influence in the prioritization of requirements, or how important their opinions are when designing requirements. Depending on their level of participation, and their role and involvement in relevant use-cases, their influence with requirements can vary from giving their opinion, to collaborating in making design or prioritization decisions.

When working on projects with unclear requirements, the core DHIS2 team sometimes include expert users, familiar with the particular use-case. These projects are usually from a stakeholder who wants support a particular use-case but doesn't know the steps required to achieve it. The core DHIS2 team then work together with these expert users to design the initial requirements to support the use-case, leveraging their knowledge about the use-case.

Donors

Donor organizations fund the development of the DHIS2 software. The funding is either directed at the general development of the software, or through externally funded projects.

Externally funded projects usually have specific use-cases they want to support, or requirements they need to be implemented. In some cases, the requirements from these projects align with existing requirements for the DHIS2 software, but in other cases, these projects introduce entirely new requirements. As a result, these externally funded projects can profoundly influence the process of prioritizing requirements.

Different externally funded projects have different timelines for when the core DHIS2 team needs to fulfill them. An important part when prioritizing requirements is, therefore, to balance which requirements to implement, and when to implement them. A vital aspect of how the core DHIS2 team is working with these projects is that the core team manages the requirements themselves, without direct involvement from the donors. By shielding the process this way, the core team themselves can decide how to work with each requirement and how they want to balance the needs of donors with the needs of other users.

Non-Donor Users

Since DHIS2 is a free open source software, anyone can use it free of charge. Especially advantageous is it for countries or organizations who do not

have the means to pay for similar proprietary software or develop their own. Although any one person, in theory, can be a free user, we will mainly discuss the most significant free users of DHIS2, the ministries of health in developing countries.

The ministries of health can use and host the DHIS2 software at different scales, from district-scale pilots to a national scale implementation. To be able to maintain and improve these implementations, they need their requirements to be supported by the software. Often, their requirements are resolved through the implementation of other generic requirements, but some of their requirements are more specific. These requirements can be as specific as support for particular types of calendars, or the ability to create reports that support their routines. Because they lack the monetary means to influence the process of requirement management the same way donors are, their influence bases itself on other values.

Free users in general, but also ministries of health, is a fundamental part of the success of the DHIS2 software. Without these users adopting, promoting and requesting the software, both new adopters of the software, and donors, and funding would be much harder to come by. The core DHIS2 team knows this, and because of this, they have a high motivation for supporting and improving the software for free users as well. In addition to the motivation of supporting and improving the software for free users, the requirements raised by these users often involves changes that benefit a significant portion of the users.

Motivated by requirements that benefit the majority of users, and to support and improve the software for both new and existing implementations, prioritizing requirements from free users is important for the core DHIS2 team. As a result of this, the core DHIS2 team makes a significant effort to balance the prioritization of requirements from free users and donors.

As mentioned previously, ministries of health are arguably one of the most significant free users in the DHIS2 community. With the growing number of use-cases and applications for the DHIS2 software, the number of requests for different requirements quickly becomes challenging to manage. It especially makes understanding the impact, importance, and demand of each request difficult.

An initiative started by the core DHIS2 team, RCAT, attempts to mitigate these challenges. The goal is to create a list of the most in-demand or useful requirements in the eyes of the ministries of health, highlighting these for the core DHIS2 team. The ministries of health do not usually interact directly with the core DHIS2 team but work with their local HISP nodes who mediate their needs to the core DHIS2 team. That means that the core DHIS2 team, together with the local HISP nodes compile this list of requirements.

5.3.2 The Core Software Developer Team

In this section, I describe the different roles of people working in the core DHIS2 team. In general, everyone influences the requirement management

Figure 5.2: A table representing the organisation of the developers in the core team.

process in some way, either through gathering, defining or prioritizing requirements. Figure 5.2 represents the developers of the DHIS2 core team, who is most involved in the requirement management process. Roles not included in the figure generally work more with gathering requirements, than generification, prioritization, and implementation. All of the roles described in this section includes their primary responsibility and impact in the requirement management process.

Developers

There are over thirty software developers in the core team of DHIS2. They are located at the University of Oslo, or remotely around the world in countries like The United States, Vietnam, Spain, Netherlands and more. The developers are all part of a different technical team, indicating what part of the software they work.

The technical teams are the Backend, Analytics, Tracker, Apps, UX, and QA team. As illustrated in figure 5.2, each team consists of 1-10 members including a team leader, except for the QA team, where the Release manager act as the team leader, and the UX team with only one member. In addition to these teams, other teams are working on the official DHIS2 android apps. Although these teams are also an important part of DHIS2 project, they are skipped in this thesis because they do not work on the core DHIS2 software itself.

Some of the developers also have extra responsibilities in addition to software development, like the lead developer, the team leaders, the release manager, and some product managers. These roles are described later in this chapter.

Developers without added responsibility, mostly influence the require-

ment management process with technical input and by estimating their work capacity. Developers are also in direct contact and discussions with expert users or attending academies to improve their understanding of use-cases and requirements.

Team Leaders

The team leader of each team assigns work, supports the team, interacts with other teams and follows up the team's members. The only exception being the Backend team, which have the lead developer as well as two team leaders. One of these team leaders is located in Vietnam and is managing team members located in Vietnam and remote developers with particular work tasks. The other team leader is located at the University of Oslo and is managing the rest of both the local and remote team members, in collaboration with the lead developer. Additionally, the team leader of the Apps team is also filling the role as a product manager together with the lead developer.

The team leaders attend weekly product meetings where they discuss both their teams capacity as well as technical details of requirements. Unlike other developers who also participate in these meetings, the team leads are usually involved in discussing all the requirements, while other developers discuss more specific requirements. These discussions focus on evaluating if the requirements are in the scope of the software, generification of the requirements, and finding and understanding technical solutions for them.

Lead Developer

The lead developer is a member of the Backend team when it comes to the software development, but is in general, interacting with all developers. They act as a decision-maker for the most significant changes. In addition to overseeing the development as a whole, they also manage the team leaders and support individual developers to some degree. The primary responsibilities of the lead developer include having an overview of the current development across teams, the recruitment of new developers, and working with stakeholders to define requirements and solve issues.

In addition to this role, the lead developer in the core DHIS2 team also inhabits the role of assistant product manager for the Apps product team. The lead developer has significant impact on the requirement management process, like deciding technical solutions, rejecting out-of-scope requirements, and more. Additionally, the lead developer also supervises the significant changes and efforts made to the software.

Product Managers

Product managers, also known as product owners, are unique roles that manage a specific part of the DHIS2 software. Each product manager leads a product team, whose members can be implementation advisors

or a combination of developers from different technical teams. While each of the technical teams relates to a specific technical area, the product teams focus on one of the core features of the software. Conveniently the product teams are split into the same grouping of people as most of the technical teams: Analytics, Tracker, Apps, and Android. The product managers are responsible for arranging weekly product meetings, reviewing requirements, understanding their respective product and the needs of the stakeholders. Together with their product teams they also plan roadmaps and prioritize requirements for implementation. The product managers play a central part of the requirement management process, acting both as a gatekeeper for new incoming requirements, but also as a source of knowledge about what users need and the future of the product.

In theory, all requirements go through the product managers before they present them to the product team. In reality, minor changes and requirements often skip this process, usually approved by a team leader or the lead developer. In cases where requirements are involved or a need a significant amount of work, the product manager evaluates the requirements and prioritize them, then brings them to the product team. This process is both done by the product managers themselves, or in collaboration with other product managers because the more complex and elaborate requirements can affect multiple products.

On a weekly basis, the product manager arranges a product meeting, attended by the product manager, implementation advisors, and developers. During these meetings, the team reviews a list of requirements presented by the product manager, identifying possible overlapping requirements or existing features, discussing technical solutions and any cross-team dependencies. After reviewing and accepting a requirement, the team prioritizes it relative to the rest of the requirements on the list, and a developer is assigned to work on the requirement.

Release Manager

The release manager was a role recently introduced, taking over the responsibilities related to releasing new versions of the software, which was previously managed by the lead developer. The responsibilities of the release manager include a variety of activities, like updating demo instances of the software available online, compiling new versions of the documentation and writing release notes.

In regards to the requirement management process, the release manager works with the different product teams and evaluates the progress of different tasks leading up to a release. Any tasks that are incomplete and risk delaying the release will be identified and brought to the product teams to discuss. This discussion will decide whether to move the task to the next release, if resources need to be reallocated to finish the task, or if the delaying the release is necessary.

In addition to alleviating some of the release-related responsibilities from the lead developer, the purpose of the release manager is to stabilize the software releases. There are primarily two challenges the core team

struggled with that caused the need for this focus on stabilization.

The first challenge is related to the stability of the software itself. Previously, although the core team invested significant time in testing the software before a new release, new releases suffered many problems with bugs. Not only did the bugs affect new features, but sometimes also existing features. The release manager organizes and plans more efficient testing of the software before releases, including both testing by the core team testers, developers, but also members of the community. Additionally, the integration of routines related to testing in the development process itself has reduced the need for extensive testing before a release, allowing for the team to focus their efforts where it is most effective.

The second challenge was related to the repeatedly postponing of releases. Due to the increase in the number of requirements done each release, and having most of the testing efforts right before a release, releases would often have to be delayed. The release manager, together with other members of the core team, worked to design a new development process. This new process is much more structured than previously, dividing the time between releases into several periods. Each period is in theory separate from other periods, very similar to how traditional agile sprints work. By planning and reviewing tasks for each period, the release manager can work with the different product teams to identify any risks and resolve them long before the release.

The influence of the release manager on the requirement management process is limited, but by identifying risks, he provides feedback to the product managers. This feedback can then be used to re-evaluate their prioritization if needed or delay the release if the delayed requirements are critical.

Project Coordinators

DHIS2 is funded by externally financed projects and contracted work. The three project coordinators in the core team are the ones who work with the donors to define and follow up requirements, as well as write and sign contract with them. The project coordinator role is often that of a mediator between the core team and the donors.

In the case of requirement management, the project coordinators represent the donors. As such, project coordinators have significant influence in the process. Both in prioritization, to making sure that requirements are implemented as promised concerning timeliness and functionality, as well as notifying donors about any issues or delays that might occur.

Researchers and Students

Master students initially developed DHIS2, and still today some contributions to the software core are the results of the work of students and researchers. Today, the researchers and students themselves have little direct influence in the requirement management process, but in 2019 a new

role for research evaluation and coordination will be created and filled. This person will work on linking research and development, providing more practical results from research and evaluation of implementations for product managers and developers to use.

Implementation Support

Implementation support refers to a group of different roles that in some way support implementations or the community. The responsibilities and areas of work of these roles both overlap with each other in some ways and are very different in other ways. There are three types of implementation supports staff: global, regional and HISP UiO implementation support staff. Global implementation support staff and HISP UiO implementation support staff are working for and employed by HISP UiO. Regional implementation support staff are usually partly funded by HISP UiO and partly funded through sub-contracts with donors.

These staff, to some degree, all work with implementation support, capacity building and defining requirements. That includes activities like managing the Discourse platform, organizing academies, creating training material, training in academies, and more. Some of these people also have multiple roles, like product managers or project managers.

Some of the implementation support roles are also more specific to the type of support they provide, like the health implementation advisor and the technical implementation advisor. The health implementation advisor has a health/clinic background and helps design requirements around the tracker-module of DHIS2 and specific use-cases for it. The technical implementation advisor provides similar value to the core team, but with a focus on the technical aspects of requirements and use-cases.

Finally, there is an implementation coordinator. In addition to the responsibilities of supporting implementations, also oversees the other implementation support staff.

Implementation support staff generally don't participate directly in the requirement management process, unless they do so in the capacity of project managers. Instead, they work with developers and product managers directly, depending on the complexity and severity of their requirements. In other words, they influence the process by working directly with the developers or product managers, by providing insight into the needs of implementations.

5.3.3 The Requirement Management Process

In this section, I describe the requirement management process of the DHIS2 core team as outlined in figure 5.3 on the following page. The table consists of the three stages generification, prioritization and implementation, each listing methods and stakeholders involved in the respective stage.

	Methods	Stakeholders	
Generification	Combine similar requirements Identify existing overlapping features Remove domain specific information Discuss requirements with stakeholders Discard out-of-scope requirements	Community members* Product managers Developers**	Requirements are discussed over one or more meetings, where similar or overlapping requests are identified and merged. A lot of requirements are domain-specific and is stripped of any domain specific information. Any requirements that is too big, or falls outside of what is considered the scope of the software is discarded. Community members are included in discussions for more elaborate requirements.
Prioritisation	Identify contracted requirements Overall demand from community Overall benefit for users Future vision of the software Core team capacity	Project participants*** Product managers Team leaders Lead developer	Prioritising requirements are done within the core team, taking multiple variables into consideration. Requirements specified through funding contracts are the most critical. Requirements are also prioritised on how well they conform to the future vision of the software, the overall demand from and benefit for users, as well as the available capacity of the core team.
Implementation	Implement features Testing features Documenting features Presenting features Including feature in training material	Developers In-house testers Expert users Product managers Academy facilitators and training material creators	After a feature has been implemented, the feature is documented by the developer and tested by both in-house testers and voluntary expert users. New features are presented initially in-house for the core team, as well as later in academies. Training material is also created and included in academies relevant to the feature.

* Community members here refers to members identified as owners of the features, usually expert users or representatives from donor organisations
** Developers here refer to developers, team leaders and lead developer
*** Project participants refers to members of the core team who is heavily involved in a project. This can include project coordinators, implementation coordinators/advisors and developers, depending on the project.

Figure 5.3: A table describing the methods and stakeholders for each stage of the requirement management process

Requirements

Gathering requirements for DHIS2 happens in a variety of different ways. In addition to users in the community submitting new requirements to support new or existing use-cases, the core team also has people actively gathering requirements. Implementation support roles actively work on understanding the existing and new use-cases of DHIS2, usually working closely with the users through implementation support or through DHIS2 academies. These people gather and define requirements by identifying new needs and gaps to bridge to improve or add support for use-cases. Also, organisations funding DHIS2 development, collaborating closely with the project coordinators, work with the core team through externally funded projects to add support for specific use-cases. Another process for gathering and understanding the demand of requirements is RCAT (Roadmap Country Advisory Team). RCAT is an effort by HISP UiO to understand the needs of the ministries of health more efficiently. HISP UiO together with HISP partners representing the local ministries of health, collaborate to make a list of requirements that would be most useful for their implementations.

When the core team receives requirements, they are added to the online issue-tracker either described in broad strokes for smaller issues, or with a more abstract description where the core team needs to work out the details themselves. The latter is usually the case when the requirements are either unknown or more complicated to solve. For example, externally funded projects might know what the final result should look like, but don't know the steps to get there. In these cases, descriptions are purposely abstract for the main requirements, then later expanded on by the core team when the specifics are more apparent.

Generification

Due to the difference in technical knowledge, understanding of their problem or experience with the software, users submit requirements in all shapes and sizes. Some users, usually expert users with both proper domain and technical knowledge, and experience working with DHIS2 submit well described generic requirements. These are requirements that with a small amount of work fits right into the software. Other users might submit domain-specific requirements, partially generic requirements, requirements with insufficient information or requirements that are well thought out but wouldn't fit into the software in its current form.

The product managers, working as gate-keepers for incoming requirements, weed out those requirements that are outside the scope of the software. Furthermore, they group similar requirements and follow up on requirements that require additional information. When the majority of the invalid requirements have been filtered out, the product managers pick a list of requirements they regard as relevant for the next version of the software. This list is the brought and presented to the rest of the product team. Together with the product team, they go through the list and discuss each

requirement, looking at them from different angles.

From a technical point of view, requirements can be invalid because there is already support for a different, overlapping requirement, or it might not be possible to implement them because of technical limitations. Additionally, the consequences of implementing the requirement are also taken into consideration to decide whether to accept it or not. These consequences include things like performance, privacy, security and best practices. The team also looks at the requirements the product manager have grouped due to their similarities, and discuss whether it is possible to find a common generic solution to all requirements, or if they need to be discarded or changed. More often than discarding these requirements, the requirements are changed to more easily fit the software or a more generic solution.

Aa a result of the variety of use-cases, as well as local needs, even generic requirements from the users are sometimes not generic enough to fit into the software. In these cases, the product team discusses different technical solutions for further generifying these requirements, so they fit not only a specific use-case or local need, but so they can be applied to other use-cases and contexts as well.

In rare cases, the core team will accept requirements that are only partly generic. That usually happens if a requirement is critical for a particular use-case and the requirement fits into the software, but will primarily only be beneficial for some users. Because DHIS2 was primarily a health information system, several design decisions have been made to support a health-specific requirement, both to make sure large implementations of the software can still use the software, but also from donor organizations with particular and critical needs.

Depending on the complexity of the requirements, a requirement can be generified, accepted and have a technical description of the implementation after one or more product meetings. When this happens, the product manager, together with the team leader and relevant developers, assigns the work to a developer and have a brief discussion to make sure all parties agree and understand the requirement. Because developers most often are part of the product meetings as well, this discussion usually happens as part of the meeting.

Prioritization

The initial stage of prioritization happens when the product managers pick out requirements they want to present to the team. During this stage, the product managers pick out requirements based on the following criteria:

- Is the requirement a deliverable? (A requirement related to an externally funded project)
- Is this requirement marked by RCAT, or needed by most users?
- Does the requirement align with the future vision of the product?

For practical reasons, the product manager also picks a variety of small and big requirements to avoid blocking all the developers with big tasks, as they often end up working on requirements from multiple product teams.

After a significant portion of the requirements have been well defined and assigned to developers in the product meeting, the meeting participants work together and discuss the prioritization of the requirements. At this point, the participants in the meeting potentially include anyone who is relevant in prioritizing the requirements, like the product manager, team leads, lead developer, project managers or expert users. The meeting participants give their opinion on the importance and significance of the requirement. Depending on each person's role or knowledge about the requirements, their opinion influences the priority of the requirement to different degrees.

From the product managers point of view, what determines the significance of a requirement is his or her knowledge about the product and its users. Requirements that would solve a common, significant or outstanding problem for most of the product's users, would be prioritized higher than a requirement that solves technical issues that might not affect the products users directly. Furthermore, requirements that the product manager recognizes as a stepping stone, or a part of the future vision of the product, might be considered more important than a requirement that solves a short-term problem.

Project managers advocate for requirements that benefit their projects. Because projects and funding are most often connected, this implies that the project manager can point out requirements that they assess as critical for the project. Alternatively, they are also able to point out requirements that are less critical for the project, and can, therefore, be prioritized lower for the benefit of other requirements. Because the core team depends on the funding from these projects, the project managers have much influence when deciding the priority of requirements.

The team leaders and developers, sometimes together with the lead developer, contribute to the discussion from a technical point of view. By assessing the complexity of requirements, the impact of the requirement on the rest of the software, and the capacity of the developers, they can provide some insight into the work required. Requirements they see as straight forward, or small might get a higher priority because fresh developers or developers who are only working on a few elaborate requirements can more easily work on them. On the other hand, requirements that are complex, and need extensive knowledge about the source code and the software, are considerably harder to prioritize. That is usually because of the combination of the time required to implement the requirement and the experience of the developer, so for complex requirements with a short deadline more experienced developers are required. Pointing out these requirements to the rest of the product team allows them to decide what requirements are more important if there are too many for the technical teams to handle efficiently. Finally, requirements that will have a more significant impact on the rest of the software, meaning a high risk of introducing bugs and a greater need for comprehensive

testing, will receive a higher prioritization so they can be implemented early in the development process, leaving a reasonable margin of time to test the changes extensively.

The final influence on the prioritization process is the RCAT initiative. Each time product managers pick new requirements to present to the team; they make sure to include a selection of requirements marked by RCAT as well. In many cases, these requirements are already requirements that are attractive to implement. That is because they often solve problems many users are facing, or introduce functionality that aligns with the future vision of the software. During the product meetings when the product team discusses prioritization, RCAT issues will more easily be prioritized higher than similar requirements, especially if multiple requirements are considered equally important.

Implementation

After assigning a requirement to a developer, the developer starts working on its implementation. While most requirements are straight forward to implement, some of those who are more complex require a more work and technical decisions before implementing them. Developers working on these complex requirements often work with other developers to find technical solutions as well as expert users to understand the problem better. Working with other developers and expert users helps them choose the most appropriate solution that both have an elegant technical and generic implementation, while at the same time solving the fundamental requirement.

When a developer completes the implementation of a requirement, they perform an initial test of their work before committing their code to the central source code repository. The next step is writing test-cases and documentation of the requirement. Both the test-case and documentation is later used by other developers, testers and expert users to extensively learn about and test the newly implemented changes before including them in the next version of the software.

When a new version of DHIS2 is released, the release manager sends out an announcement to the community using the Discourse platform. This announcement, labeled as release notes, contains an overview of the most exciting or significant changes included in the new version. The release notes include a short description of the changes, some screenshots if applicable, and a link to the documentation describing how to use the feature. This process provides users with an awareness of the new changes, so users can try them out themselves, allowing them to evaluate if they should upgrade their software to benefit from the change.

Shortly after the release of a new version, The core team presents the new and improved features in the new release to the other members of the core team. These presentations usually happen in the form of an internal training session or a seminar. Similar to the release notes announcement, these presentations are limited to the most significant changes, like new features, significant changes to the software or that

significantly change how users interact with the software. Significant changes include changes like increasing the required versions or adding required third-party software, new apps or changes in the data models.

By first educating the core team, they can help identify which features can be challenging for users to grasp, which requires more documentation or even uncovering unexpected applications for a feature. Especially the latter benefit is essential in managing the requirements in DHIS2 because by recognizing different uses of a feature we can identify requirements that would potentially overlap with this feature. Recognizing overlapping features allows us to reuse existing functionality in the software, or add support for functionality with less effort by extending the current feature, as an alternative to implementing a brand new feature. Because the core team often work closely with the expert users of DHIS2, there are several instances where the expert users can join these presentations as well. These expert users, working with the software in a production environment, brings an entirely different perspective than that which the core team has. Especially regarding understanding how users are interacting with the feature and the software, and how it will perform in a practical context.

Developers primarily write all the documentation for DHIS2, and it is one of the most accessible sources for users to increase their knowledge about the software. The developers are usually contributing to the documentation with features they have implemented. As a result, language consistency, the level of technical detail and description of the documentation can differ a lot from developer to developer. The varying quality of the documentation is primarily due to most developers being non-native English speakers and their understanding of how to describe a feature could be very different from what the users who are reading it needs. Each contribution to the documentation is reviewed by a member of the core team, allowing for some threshold of consistency, but for new, non-native English speaking DHIS2 users, with limited technical knowledge, learning to use DHIS2 from documentation alone can be challenging.

In addition to the documentation and community platforms, new features are both announced at the different DHIS2 academies and included in the training material. Together, the academies, community platforms, and documentation help the users increase their knowledge about new features. Some academies also include a plenary session for facilitators and users interact in a two-way fashion. During these sessions, facilitators and users often discuss both specific features of the software as well as topics more related to particular use-cases. Users can, during these sessions, also directly provide feedback on their needs to the facilitators, who sometimes include members from the core DHIS2 team or someone who has direct access to the core DHIS2 team and can act as mediators for the users.

Periods	The development of DHIS1 from the beginning to the release of DHIS1.4 (1994 - 2006)	The development of DHIS2 from the beginning until 2017. (2004 - 2017)	The most recent development of DHIS2. (2018)
Interaction with users	Singular PD initially, Serial PD after scaling out of South Africa.	A mix of Singular, Serial and Parallel PD.	A mix of Singular, Serial and Parallel PD.
Core Developers	• 2 in South Africa	• Master students and teachers at UiO, Norway, until 2011. • From 4 to 30 developers between 2011 and 2017. • 4 to 26 developers working from UiO, Norway. • 4 working remotely from U.S and Vietnam.	• 39 developers. • 20 developers working from UiO, Norway. • 19 developers working remotely from U.S, France, Vietnam, Germany, Spain, Sweden and Netherlands
Known implementations of DHIS2 in Ministries of Health	• National implementation in South Africa. • Multiple pilot projects in other countries.	• 45+ national implementations. • 10+ full-scale Indian states. • 15+ pilot projects.	• 50+ national implementations. • 10+ full-scale Indian states. • 15+ pilot projects.

Table 5.2: A summary of the different periods of DHIS development

5.4 Summary

Both the development of the DHIS1 software and the DHIS2 software, have a lot in common in the way they have been managing requirements. The fundamental ideas behind software development are very similar, like participatory design and having an open development process where users can contribute.

The core DHIS2 team attempts to design the software closely with its users, as the developers did during the development of the DHIS1 software. While the growth of DHIS2 regarding the number of users and use-cases has introduced several challenges in that regard, the core DHIS2 team creates several opportunities to accomplish this kind of interaction with the users. These opportunities include collaboration through the community platform, like Discourse, and through academies. Figure 5.2 on the preceding page lists the different types of participatory design used in the different periods.

Both the development of the DHIS1 software and the DHIS2 software have experienced different challenges in their lifetime. The challenges in the DHIS1 development are mostly related to the software itself, for example, being unable to transfer the software to other countries easily. The DHIS2 software, on the other hand, faces challenges related to its growth, including dealing with a bigger core team, and several different use-cases and contexts to support. Table 5.2 on the facing page gives an overview of how the core teams, the number of implementations, and the interaction between developers and users have changed through the different periods.

Although being based on the DHIS1 software, the DHIS2 software has evolved into something much more than a simple HMIS system for the health domain. It supports several new use-cases, both in the health domain and other domains. One of the most significant differences from the DHIS1 software is the "Tracker" module. This module allows users to record data on an individual level, in addition to the traditional recording of aggregated data.

With this evolution, the DHIS2 software has diverged significantly from the DHIS1 software, and as a result, deals with a completely different set of challenges. Designing generic features that transcend not only different use-cases but also different domains, and balancing the needs of the free users and the needs of those providing funding, are examples of challenges seen in the development of the DHIS2 software today.

The DHIS2 software has an extensive eco-system built around it. This eco-system including online tools for collaboration, academies for capacity building and HISP nodes which help mediate the users needs to the core DHIS2 team. Through this ecosystem, the core DHIS2 team can support users, gather and design requirement, learn about the different use-cases and more.

Chapter 6

Discussion

In this section, I apply the framework introduced in chapter 4, summarized in table 4.1 on page 24, to the different periods described in chapter 5. These periods, which is listed and described in table 5.2 on page 48, combined with the framework, let us describe the requirement management process for each period in a structured way. This structured description of the process allows us to more easily compare and discuss the changes made to the process and organization in the different periods.

I will also describe some of the known challenges the core teams faced during each period in regards to the requirement management process. For each challenge, I also discuss the reason for them, and potentially how to deal with them.

6.1 DHIS1: 1994 - 2006

The development of the DHIS1 software included a relatively small number of user in South Africa. The two developers working on the project worked closely with the users in a singular participatory design fashion. This approach was advantageous and allowed the developer to rapidly release new versions of the software, and provide great support to the users at the same time.

When scaling to other countries, it became difficult to provide the same face-to-face singular participatory design interaction the core team of developers had previously used. With frequent trips to the new implementations, and with the help of implementation coordinators mediating between the developers and users, the developers managed to get the implementation up and running. This approach to user interaction is arguably more similar to that of serial participatory design. That means the development of the DHIS1 software utilized a combination of both the singular and serial participatory design methods.

There is not much information related to the development concerning generification in the early versions of the DHIS1 software. However, because the developers were working with the DHMT in South Africa, we can argue that they provided requirements designed to support all potential implementations in South Africa. That means generification was

partly done by the user, or community, by utilizing their understanding of the different requirements. Additionally, we know that users gained access to the core team based on their merit, implying they could more easily influence the design of requirements than those users with less merit. Their process of generification can be said to include methods similar to the "Management by Community" and "Management by Social Authority" methods.

Based on the description of how the DHIS1 software was developed in collaboration with the users, the project had a governing method similar to the Bazaar method. It was open for all users to participate and users were granted access in a meritocratic fashion. One significant difference, however, is that there doesn't appear to be possible for any user to gain any form of leadership or special authority based on their merit, because the developers made all final decisions.

The DHIS1 software development was externally funded, and the software was free to use, although it required proprietary software to work. This model of innovation is similar to both the collective-action and the private-collective models. Because some participants invest their resources, namely the organization funding the project, it is mostly similar to the model of the private-collective.

Types of Participatory Design	Singular, and later Serial
Methods of generification	Management by Community, Management by Content, Management by Social Authority
Method of Governance	Bazaar
Model of Innovation	Private-Collective

Table 6.1: Summary of DHIS1 between 1994 - 2006

Table 6.1 gives a summary of the DHIS1 software development in this period, described using the framework found in table 4.1 on page 24.

6.1.1 Challenges

What seems to be one of the core challenges for DHIS1 is related to the software itself. Although the software was a perfect fit for South Africa, it was initially not flexible enough to be adopted in other countries.

First of all, the software was designed closely with the users in South Africa, imprinting their ideas into the software itself. Unfortunately, this could not always be transferred to other contexts. For example, due to political reasons in Cuba, where they wanted to avoid empowering the users and have all decisions coming from the top. Because the software was designed with the idea of empowering users, to be able to move the software to these contexts, the fundamentals of the software would have to change. One can argue that this challenge of moving the DHIS1 software to new contexts is the result of an insufficient generification of the requirements. Although the design and decisions made in South Africa

were appropriate at the time, the requirements were not generic enough to easily be transferred into other contexts. Because the primary objective of the project was to develop an HMIS for South Africa, trying to account for requirements outside of this scope would probably not have been the correct decisions at the time.

Secondly, the source code of the DHIS1 software had become "messy" and difficult to maintain. As a result, some changes required significant effort to implement. The "messy" source code is likely the result of the rapid prototyping of DHIS1 and active collaboration with users in South Africa. When developing software and making several changes and improvements over time, source code can quickly end up in this state. The implication of having the source code in this state is that supporting the requirements of these new implementations became difficult. Making new changes to the source code of the DHIS1 software required rewriting existing code to allow to the changes to be made. The core developers had to do major changes in the source code to be able to add the changes needed to support new implementations, in what would become DHIS1.4.

Finally, because the core developers of DHIS1 was located in South Africa, it proved troublesome to provide the same level of user interaction with implementations outside South Africa, that they had before. Compared to working with the South Africa implementations, working remotely like this seems to have slowed down the process because of the need to travel and exchange information.

Some of these challenges would not be solved by the core team themselves, like the incompatibilities of the software due to political reason. However, regarding the support of new implementations, they used implementation coordinators to mediate between the remotely located implementations and the developers in South Africa.

6.2 DHIS2: 2004 - 2017

The core DHIS2 team interacts with the users in several different ways when gathering and designing requirements. For example, requirements are gathered implementation support staff, expert users and developers, either directly with implementations, or through academies.

The interaction with users in regards to gathering and designing requirements done through the work of implementation support staff, or DHIS2 academy participation, is similar to Serial PD. It could arguably be described as being similar to Singular PD as well, but from the perspective of the core DHIS2 team, it is more accurate to describe it as Serial PD.

Through the customization of the DHIS2 application or development of custom apps, a similar type of interaction occurs. These activities are usually an interaction between HISP nodes or external developers, and local implementations of DHIS2. The collaboration between these parties could be described as Singular PD. However, in regard to the requirement management process of the core DHIS2 team, it is that of a Parallel PD, because the interaction is not by the core team themselves.

Although the core DHIS2 team sometimes travel to users and work closely with them, it's not a sustainable way to design requirements. At the University of Oslo, the product managers, together with other members of the core team, filter, review and generify requirements gathered from the community. This approach is most similar to that of the Parallel PD. That means that at different levels, and to varying degrees, the DHIS2 software is developed using a mix of all three types of PD.

Requirements are gathered and designed in several ways, as described above, and by different types of people. As a result, several methods are applied when attempting to develop generic requirements. For instance, during plenary sessions in DHIS2 academies, participants learn about each other's use-cases and needs, allowing them to work out requirements that solve problems that are common for all of them. Additionally, having expert users participating in these sessions can help the process by understanding what requirements are outside of the scope of the DHIS2 software and what problems can be solved by aligning their processes to those of similar use-cases. In other words, both generification methods from the management of community and the management of content are used during academies.

The core DHIS2 team at the University of Oslo also apply the similar generification methods when working with requirements in their product meetings. For example, management by community methods by working with HISP nodes and implementation support staff. The HISP nodes and implementation support staff work with several users and implementations from the community and gather requirements that are relevant to the community. When the core DHIS2 team reviews these requirements, they have already partly been generified by the community through the HISP nodes and implementation support staff. That means the community has the responsibility of generification in many cases, at least to some degree.

Additionally, through the requirement management process of the core DHIS2 team, several of the management of content methods are applied. For example, methods like process alignment, comparing different use-cases, identifying organisationally particular requirements and which requirements are outside the scope of the DHIS2 software.

There are also cases where the core DHIS2 team also uses the management by social authority methods. For example, some stakeholders, like donors and HISP nodes have more influence when designing requirements, as they represent both the source of funding for the core DHIS2 team, but also some of the most significant implementations of the DHIS2 software.

The way the DHIS2 software has been designed, allowing for extensive customization of the software and development of custom apps, enables users to make local innovations. This form of open generification is already happening in the DHIS2 software. For example, a custom dashboard app was created by users in the community, solving some of the limitations of the existing dashboard application included in the DHIS2 software. The core DHIS2 team reviewed this custom app, and through a process of

disembedding, they designed a new generic dashboard app, implementing some of the new useful functionality of the custom app into the design.

HISP UiO makes all final decisions about the design and future of the DHIS2 software. However, these decisions are not only based on the wishes of HISP UiO, but rather the needs of the community and the overall good of the DHIS2 software. In other words, the governance style of the DHIS2 software is more similar to a bottom-up-distributed approach, than a top-down-centralized approach.

Additionally, users can gain limited authority, in the form of the ability to influence the design and prioritization of requirements, based on their contributions. For example, expert users have a higher influence on design and prioritization than a fresh user would have. That implies there exists a meritocratic system for users to gain authority.

Finally, stakeholders outside of HISP UiO are not allowed to control the requirement management process, which means they are unable to drive their requirements into the software. For example, if a stakeholder providing funding could control what requirements would be implemented, the process would be governed with a top-down-centralized approach where participation was limited to those providing funding.

Based on all these descriptions of the decision-making process used in the DHIS2 software, we can recognize it as the Bazaar method of governance.

All contributions to the DHIS2 software, including the source code, documentation and training material, is provided freely. Even contributions from externally funded projects benefit all users of the software without any cost. As such, the Private-Collective model of innovation is a good fit for the DHIS2 software. However, innovations created by people outside of the core DHIS2 team are not necessarily provided freely to everyone. These innovations can include services or development of custom apps, which the creators might choose to keep for themselves or charge a fee to provide them. Because this is a by-product of the DHIS2 software and not a part of the software itself, it doesn't affect the model of innovation of the DHIS2 software.

Types of Participatory Design	Serial and Parallel
Methods of generification	Management by Community, Management by Content, Management by Social Authority, Open Generification
Method of Governance	Bazaar
Model of Innovation	Private-Collective

Table 6.2: Summary of DHIS2 between 2004 - 2017

Table 6.2 gives a summary of the DHIS2 software development in this period, described using the framework found in table 4.1 on page 24.

6.2.1 Challenges

During this period, the core DHIS2 team faced several new challenges, mostly as a result of their growth. For example, tracking and designing requirements became more complicated, releases were more often delayed, and the software would often have stability issues with new features. During this period, the number of developers, use-cases and externally funded projects increased, until the point where the current organization and processes of the team could not keep up.

With the increase in use-cases and externally funded projects, the number of requirements increased significantly. To keep up with the demand, the development process became something the developers at the time referred to as a "feature-factory." A "feature-factory" can be interpreted as a state of development where most of the efforts are put into new features at the cost of the maintenance and stability of the existing source code. Although this was not the intention, or sometimes not the reality for most developers, it still describes how the development was affected at the time. Developers worked on implementing requirements, like new features, to meet the expectations of the community, sometimes at the cost of spending time maintaining the software. This development trend was also most likely a result of the new developers joining the projects as well, who had less experience and knowledge about the software and the use-cases.

This challenge affected the development in several ways, like managing requirements, delayed releases and software stability as mentioned earlier. Each of these problems was dealt with in different ways. For example, by dividing the developers into separate technical teams, each with a team leader following up and reviewing work before releasing it, made it easier to validate the quality of the work. Similarly, team leaders would primarily manage requirements related to their teams themselves, making it slightly easier to handle all the requirements.

Another one of these problems, related to the externally funded projects, was to follow up, track and managing the deliverables in these projects. To deal with this challenge, HISP UiO hired project coordinators to work with the different donors and projects. This change made it easier for the core DHIS2 team to have an overview and plan development activities related to these projects.

The frequency of releases was also changed to deal with the problem of delayed releases. However, this had little impact. With less frequent releases even more changes were made to the software, and the delays and stability weren't significantly improved. Another change to deal with this issue was changing the workflow of the developers, focusing more on the maintenance stability of the software. Although this change appears to have had some effect on the problem, it was not until later the workflow had changed enough to make a significant impact.

6.3 DHIS2 today: 2018

From the previous period to this, little has changed in regards to the framework. The same mix of PD types occurs at different levels and in different contexts, and even though the organization and processes changed a lot through 2018, the underlying activities are still the same. If anything, these changes confirm the description made for the previous period.

For example, the DHIS2 mailing-lists were public and arguably assisted in the process of generifying requirements for the core DHIS2 team through the management of community methods. In 2018, with the introduction of the DHIS2 Discourse platform, this form of interaction was significantly simplified for both the community members and the core DHIS2 team. The forum inspired format itself is a much more effective way to organize discussions about different topics and to moderate the content, making it easier to find information. Additionally, the Discourse platform allows users to vote, share and cross-reference posts, making it easier to identify users who make significant and valuable contributions. These features both indirectly and directly grant authority to active users regarding recognition in the community, but also the form of additional privileges on the discourse platform. In other words, we can argue that there is an actual meritocratic system for users, which again confirm that the DHIS2 software has a Bazaar type approach to governance.

If we look past the framework, there were significant changes made to the organization and to the requirement management process itself. These changes were primarily to further deal with the challenges described in the previous period, like delayed releases, the stability of the software and the growth of the core DHIS2 team. For example, the introduction of the release manager helped reduce some of the volatility of the development workflow, by intermittently reviewing the process of each team to identify any potential risks.

Types of Participatory Design	Serial, Parallel
Methods of generification	Management by Community, Management by Content, Management by Social Authority, Open Generification
Method of Governance	Bazaar
Model of Innovation	Private-Collective

Table 6.3: Summary of DHIS2 in 2018

Table 6.3 gives a summary of the DHIS2 software development in 2018, described using the framework found in table 4.1 on page 24.

6.3.1 Challenges

Some of the problems presented in the previous period are still relevant in 2018. These problems were either not solved or only partially solved

during that period, like delayed releases, software stability and a high number of requirements to manage. Furthermore, other challenges became more apparent during this period as well, related to cross-team dependencies and a growing backlog of issues.

The challenge of having frequent delayed releases was rapidly improving during this period. The changes to the development workflow, where the time between each release was divided into smaller segments, most likely had the most significant impact. As the core DHIS2 team adapted to this new workflow, followed up by the release manager, it became easier to deal with any potentially risky issues.

Additionally, this change in the workflow also made it easier to plan and prioritize requirements. Because each release was divided into smaller periods, product managers could focus their efforts to plan one period at the time. In addition to the detailed plans for each period, product managers also have a rough plan across all periods. While overall plan support planning between different product teams, the detailed plans supported the developers. By following up both plans and reviewing the progress of developers, it became significantly easier to avoid tasks that could delay the release by reassigning or postponing them. As a result, recent releases of the DHIS2 software has fewer delays, and, when a delay is inevitable, it becomes apparent much earlier in the process. By catching the delays early, its easier to plan around the delays, and inform stakeholders.

Regarding the problems of managing an increasingly high number of issues, they have partly been dealt with but is still relevant and present today. Through the efforts of the product managers, it has become easier to deal with, but it still causing some issues concerning the prioritization of requirements and the pressure to include as many requirements as possible into each release. This challenge might be difficult to deal with completely, because the software keeps growing, adding support for more and more use-cases and increasing the number of new requirements.

One potential way to deal with this high number of requests in an impactful way would be to define and limit the scope of the software. Today, the core DHIS2 team primarily define the scope of the software in regards to the type of functionality it should or should not support. For example, the software supports some core and supportive features related to gathering, analyzing and visualizing data. Requests for features like a full-fledged task management system is therefore out of the scope. On the other hand, adding similar functionality to enhance the existing features would be accepted. The main difference of these being the scale and complexity of the feature, or if the feature could be supported through integration with other software.

However, the DHIS2 software is generic and can be used in several different new use-cases, and which uses-cases we should add support for is not restricted. One example is the Tracker module of the DHIS2 software, which initially did not support use-cases involving confidential data, like patient information. However, because the core DHIS2 team saw a trend of users requesting support for it, or even using it for this purpose

already, they decided to add the required support for these use-cases. Accepting use-cases like these, that significantly change how users can use the software generates a significant number of new requirements. If the scope was stricter, regarding the types of use-cases that can be supported, it could potentially reduce the high number of new requirements coming in.

Because the development of the DHIS2 software depends on externally funded projects, limiting the number of use-cases could impact the funding. Without sufficient funding, it is possible that the growth and success of the DHIS2 software would suffer. In other words, being flexible in what use-cases supported, and therefore dealing with a high number of requests, might be how a project like the DHIS2 software can remain successful over time.

The problem concerning the software stability is actively being dealt with, both in 2018, but significant efforts are planned after this period as well. These efforts made to improve software stability are primarily made through a significant increase in software testing.

The first effort started already in the previous period but became more significant in 2018. One of the testers in the core DHIS2 team was given the responsibility to reproduce bugs reported by the community, as well as testing changes made by the developers. Having someone validating bugs reported by the community resulted in a significant improvement to the productivity of developers. Because many bugs reported by users were the result of user error and not an error in the software, the tester could weed out a significant number of issues. With fewer issues to review, developers could focus on fixing actual bugs, allowing them to overall be more productive.

The second effort made to improve the software stability was to further include the community in the testing of new releases before they are released. Users working with large implementations of the DHIS2 software were especially valuable, as they could test the application with realistic data and on a more realistic scale. The release manager plans these pre-release test efforts and announces them to the community, making them more structured and impactful than previously.

The final effort made to improve software stability is through test automation. Although there was already a range of automated tests for the DHIS2 software previously, they only covered parts of the software. Additionally, the tests themselves were slow and time-consuming to implement, so they were mostly prioritized for the most important features. Now, several changes have been made to improve the coverage, quality, and performance of automated testing, including unit tests, integration tests, and API tests. Also the continuous integration workflow, an automated sequence of processes source code goes through when changes are made, have been improved to analyze the changes. This analysis evaluates the quality of the code and points out any potential errors, allowing the developers to make any nessecary changes.

These three efforts combined helps increase the developers' confidence in their contributions, but also help uncover a significant number of bugs

they can resolve before releasing new versions. The software stability is still an issue today, but with the continuation of these efforts, it is gradually improving.

6.4 Summary

While there are several similarities between the DHIS1 and DHIS2 requirement management process, there are also some differences that affected the different processes. Both of the projects had the same method of governance, Bazaar, and model of innovation, private-collective. Additionally, they employed similar methods for generifying requirements, except DHIS2 which also have examples of open generification being used.

The DHIS1 software initially used a Singular type of PD to gather and design requirements. However, when working on moving DHIS1 outside of South Africa, this changed into a Serial PD. The DHIS2 software however used Serial PD and Parallel PD for the same purpose. The type of PD each of these projects used reflects the number of different contexts and use-cases the software was used. For example, how the DHIS1 project started with a Singular PD when they had few, local, implementation, then changing to a Serial PD approach later to support new and remote implementations.

There are also multiple examples of open generification in DHIS2 software development, like the dashboard example in this thesis. The reason for this happening in the DHIS2 software and not the DHIS1 software could be because of the design of the software, but also because of how it has been facilitated. For example, there is a lot of effort put into capacity building for the DHIS2 software, and due to the design, it can easily travel to both new context and use-cases, which in turn provides more opportunities for local innovation.

Despite the similarities of the approach in each project, the softwares ability to move into other contexts and use-cases seems to have been the most significant difference between them. While the DHIS1 software had difficulties moving to contexts other than South Africa, the DHIS2 software is quickly supporting new contexts and new use-cases. It also seems that the more contexts and use-cases the software support, the more challenges can be found in the requirement management process.

Chapter 7

Conclusion

To answer the first research question, "How are requirements managed in global generic open source software projects?", we examined how the DHIS1 and DHIS2 software projects handled requirements, both historically and contemporary. What we found is that both projects had very similar approaches to the process of managing requirements. At the same time, the projects had a very different number of use-cases and contexts they supported. Where the DHIS1 software appears to have supported a specific use-case in a few different contexts, the DHIS2 software supports both different context and several different use-cases.

Both projects applied different types of PD when gathering and designing requirements from users. While the DHIS1 software was still developed for South Africa, they used Singular PD. However, when they started working with implementations outside South Africa, their approach changed to Serial PD. The DHIS2 software, on the other hand, uses a combination of Serial PD and Parallel PD as Singular PD is not sustainable with the number of implementations they support. It is clear that with the number and variations of contexts and use-cases the software supports, PD like the Singular PD becomes unsustainable. Even solely rely on Serial PD in the DHIS2 projects would not be sufficient to manage all the requirements in that project efficiently.

Regarding generification, both projects employ similar types of methods, except for the open generification which is only present in the DHIS2 project. Supporting multiple use-cases and contexts, combined with the software design, was most likely what allowed for the local innovations in open generification. In other words, the difference in how these projects generified requirements are probably tied to the software ability to move into different context and use-cases.

Both projects had a similar method of governance and model of innovation. The Bazaar method of governance and the Private-Collective model of innovation are both described as common for open source projects. These approaches have probably had a lot of impact on the success of the projects. Both the projects relies a lot on the users, both regarding gathering and designing requirements, but also in the case of DHIS2, their continued interest from donors, which both of these approaches have a

significant focus on empowering.

The second research question presented was "What challenges exist in managing requirements in global generic open source software projects and how can they be dealt with?". We answer this question by looking at the challenges from the DHIS1 and DHIS2 projects.

The challenge faced in the DHIS1 project was related to moving the software into new contexts. The core DHIS1 team dealt with this challenge by going from a Singular PD to a Serial PD approach to better support the new, remote implementations. Additionally, they had to make a significant refactoring of their code to be able to add the required support for these implementations.

In the DHIS2 project, the challenges were primarily related to their growth and how do deal with the increasing number of requirements for the software. With this increase in requirements, several problems followed, like delayed releases and reduced software stability. Each of these problems was dealt with in different ways, primarily by organizing developers into teams, changing their workflow and improving the focus on test automation. However, this challenge is still present today although the related problems are not as critical today. One suggestion regarding how to deal with this challenge is to limit what use-cases to support. This solution could, however, act as a double-edged sword, where the number of requirements might be reduced, but so would potentially the interest from donors, limiting the funding for the software.

The strength of the thesis is that it has a thorough description of the requirement management process of the DHIS2 software as it looks today. Furthermore, it also compares how the process has changed over time, both for itself, but also how it compares to how the DHIS1 software did requirement management. Finally, it also includes challenges related to these processes, describing some of the problems caused by these challenges as well, and how they were solved.

One of the apparent weaknesses of the thesis is the lack of comparison with other, similar, projects. Both the DHIS1 and the DHIS2 projects revolve around the same core software, and in very similar settings. Looking at other generic open source projects than DHIS could have provided a different insight into what makes the DHIS2 requirement management process different from others, or if they are similar.

The main contribution of this thesis is the rich case study of the requirement management process of both the DHIS1 and DHIS2 software. This case study provides a unique insight from the perspective of one of the DHIS2 developers. Additionally, a framework for describing and discussing these processes have been proposed, using methods and models found in existing literature.

The case study and framework can be used in other research as well. For example, by comparing the requirement management process of the DHIS projects to other projects either proprietary or open source. The case study can also work as a foundation for additional research regarding the DHIS projects. Additionally, the challenges presented could serve as problems to further investigate in other research.

Bibliography

[1] Kristin Braa and Richard Vidgen. 'Interpretation, intervention, and reduction in the organizational laboratory: a framework for in-context information system research'. en. In: *Accounting, Management and Information Technologies* 9.1 (Jan. 1999). ., pp. 25–47. ISSN: 09598022. DOI: 10.1016/S0959-8022(98)00018-6. URL: http://linkinghub.elsevier.com/retrieve/pii/S0959802298000186 (visited on 24/01/2019).

[2] Abyot Asalefew Gizaw, Bendik Bygstad and Petter Nielsen. 'Open generification'. en. In: *Information Systems Journal* 27.5 (Sept. 2017). ., pp. 619–642. ISSN: 1365-2575. DOI: 10.1111/isj.12112. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/isj.12112 (visited on 10/12/2018).

[3] *Health Information Systems Programme (HISP) - Department of Informatics*. . URL: https://www.mn.uio.no/ifi/english/research/networks/hisp/ (visited on 21/01/2019).

[4] Eric von Hippel and Georg von Krogh. 'Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science'. In: *Organization Science* 14.2 (Apr. 2003). ., pp. 209–223. ISSN: 1047-7039. DOI: 10.1287/orsc.14.2.209.14992. URL: https://pubsonline.informs.org/doi/10.1287/orsc.14.2.209.14992 (visited on 10/12/2018).

[5] *HISP Groups - Department of Informatics*. . URL: https://www.mn.uio.no/ifi/english/research/networks/hisp/hisp-groups.html (visited on 01/02/2019).

[6] Petter Nielsen and Sundeep Sahay. 'Editorial: Critically studying openness: A way forward'. In: *Information Systems Journal, accepted for publication* (2019). .

[7] Neil Pollock, Robin Williams and Luciana D'Adderio. 'Global Software and its Provenance: Generification Work in the Production of Organizational Software Packages'. en. In: *Social Studies of Science* 37.2 (Apr. 2007). ., pp. 254–280. ISSN: 0306-3127, 1460-3659. DOI: 10.1177/0306312706066022. URL: http://journals.sagepub.com/doi/10.1177/0306312706066022 (visited on 10/12/2018).

[8] Lars Kristian Roland and Terje Aksel Sanner. 'P for Platform. Architectures of large-scale participatory design'. en. In: 29 (2017). ., p. 33.

[9] Sundeep Sahay and Jørn Braa. *(PDF) Integrated Health Information Architecture: Power to the Users*. en. . URL: https://www.researchgate.net/publication/314263419_Integrated_Health_Information_Architecture_Power_to_the_Users (visited on 01/02/2019).