UNIVERSITY OF OSLO
Department of informatics

Explore the challenges of providing documentation in open source projects

Master thesis
60 credits

Margrethe Store

July 31, 2007

Abstract

It is well known that software documentation in open source projects is often poor and incomplete. Open source communities are generally driven by project members doing what they want to do, and because few programmers enjoy writing documentation, many open source projects are poorly documented compared to proprietary projects. This does not mean that documentation is any less important in open source projects, and this thesis looks at why it is so hard to provide good documentation. Findings from this thesis shows that even if all project members agree that documentation is important, resource constraints mean that the time and effort necessary to create quality documentation it is not necessarily provided.

How lack of documentation is affecting new project members who try to contribute to a project is also described in this thesis. Several new project members found the given documentation to be messy and outdated, making it hard to contribute. Poor documentation can also influence the number of project members willing to contribute to the open source project.

The thesis is based on an action research project where the author has participated in the development of a health information system, District Health Information System version 2 (DHIS 2), within the Health Information System Programme (HISP) network.

Acknowledgements

I would like to thank my supervisors Knut Staring and Ola Titlestad for guidance in writing this thesis and support during my time in Vietnam.

I would like to thank the Vietnamese HISP employees in Ho Chi Minh City and Hue for their friendship and the warm welcome they gave me to their country. I would also like to thank my fellow Norwegian students who were with me during my time in Vietnam. Your support and friendship has been of great value.

I would also like to thank the rest of the DHIS 2 development team for great collaboration, for answering my questions and providing support when needed.

A warm thanks to my friends and family who have supported me and put up with me during the time of writing this thesis. I would like to give a special thanks to Are for proofreading and believing I could do this. Your support has always helped me.

Table of Contents

List of figures

List of tables

1 Introduction

This thesis is based on the development of the District Health Information Software version 2 (DHIS 2) that is a part of the Health Information Systems Programme (HISP). The author has been part of the DHIS 2 development team at the University of Oslo with a field trip to Vietnam from July to November 2006.

1.1 The action research project

I have been involved in the development of a global open source health information system called DHIS for the HISP project. HISP is a global research and development network aimed at improving health information systems for developing countries. DHIS 1 was implemented using Microsoft Access and Visual Basic. This thesis focuses on the development of the second version of this software, DHIS 2. DHIS 2 is an open source web-based Java application, developed using open source tools and frameworks. The development of DHIS 2 is distributed among four collaborating nodes located in Norway, Vietnam, India and Ethiopia. Since I am situated in Norway and have been on a field trip to Vietnam for four months, this thesis is mainly focused on the Norwegian and the Vietnamese node. India is also an important part of the general development effort and is therefore frequently made references to.

1.2 Motivation

I was introduced to the HISP project and the DHIS software through a course at the University of Oslo called "Open source software development and Java frameworks in global networks", which was held by one of the coordinators of the HISP project. The focus of the subject was to learn about the goals of HISP, help with the development of DHIS 2 and learn about open source software in general. The course gave me an introduction to the tools and frameworks used in the development of DHIS 2, and I got acquainted with some of the developers based in Norway. Java has always been a favourite programming language of mine, and I was enthusiastic about the opportunity to learn more about Java-related tools and frameworks.

My motivation for writing this thesis originates from personal observations made as a member of the DHIS 2 development team. I have actively used, favoured and been interested in open source software for several years. The HISP project gave me the chance to learn more about this exciting topic and actually participate in an OSS project.

I have often felt that documentation is neglected in open source projects, and experienced this in the DHIS 2 project as well. Being, for the first time, a participant of an open source software project, I decided to take the opportunity to investigate the documentation in detail.

One of the HISP goals is to provide better health information systems (HIS) to marginalised countries. From an ideological point of view, I deeply believe open source software and information and communication technology can have a positive impact on poor countries and communities. To work with HISP gives me a chance to visit one of these countries and potentially enables me to make a difference and contribute to improving the welfare of the citizens in that country. This is an exciting and highly motivating prospect.

Additionally, working on software that is deployed and used in real life is a great way to learn more about every aspect of the software development process and a very valuable experience for me as a software developer.

1.3 Research objectives

Primary research objective

Explore the challenges of providing documentation in open source projects.

By documentation, I am referring to any artifact whose purpose is to communicate information about the software system. These artifacts can be end user documentation, manuals, software documentation, both in the source code and external documentation, mailing lists and general knowledge sharing within the project.

Open source projects are typically organised in a distributed and decentralised manner, and these factors strongly influence the development processes and the type of tools that can be utilized (Erenkrantz and Taylor, 2003). Globally distributed projects have to deal with many problems arising from participants not speaking the same language or being in the same time zones, participants having different work ethics and hardware and software requirements, plus cultural differences in general.

By taking part in the development of the health information system DHIS 2, and by being part of that development community, I will explore the challenges of providing documentation. To do this I will look at the documentation written before I joined the project and the other tools and technologies being used in the project which may have an impact on the documentation and the writing of it.

Secondary research objective

Investigate how lack of documentation affects new project members.

Goldman and Gabriel (2005) state that it should be as easy as possible for new developers to learn their way around the source code. As being new to an ongoing open source software project I want to see which impact documentation have on participants, and especially new project members who decide to join the project.

As I explore the problems caused by a lack of documentation, I will also discuss related knowledge sharing issues. I have approached these objectives through an action research project and will draw on my experiences from this process when I explore the research objectives.

This thesis covers the common issues found in open source projects after they are founded and does not discuss the establishment of open source projects. Earlier research about the initial phase of the DHIS 2 project has been conducted by Nordal (2006). Former research about DHIS 2 is described further in chapter 2.4.

1.4 Structure of this thesis

This thesis is structured into four parts and 7 chapters. Each chapter opens with an introduction to the included contents. The following parts are presented:

- Literature and Background – the theoretical framework for this study and former research on DHIS 2 is described in chapter 2. Chapter 3 tells the history of the HISP project and the DHIS software. The development process of DHIS 2 is also described.
- Methods – the research approach used in this thesis is presented in chapter 4.
- Empirical study – the empirical material used in the thesis, focusing on my experiences from Vietnam, is presented in chapter 5.
- Discussion and Conclusion – chapter 6 and 7.

2 Literature review and background

In this section I will present the theoretical background relevant to my project. Theories and strategies outlined here will be reflected in my empirical study and then discussed in relation to my empirical findings.

2.1 Open source

Computer users have been sharing software since the beginning of the computer era, and the origin of open source software can be traced back to the 50s. Back then, all software was available for free, and most of it was open so the user could examine the source code if they want to. You bought the hardware and got the software thrown in for free. It was available for free because it had not really occurred to anyone that it had value, and it was open because there was no reason for it not to be, as it had no value in the market (Glass, 2005).

The software remained freely available until the mid-60's when the hardware and software was separated, making it possible to sell software. Manufacturers started to ship software with licenses that more strictly enforced their copyrights.

In the 80s, when software was increasingly commercialized, Richard Stallmann founded the Free Software Foundation (FSF) and the GNU Project (Hars and Ou, 2001). FSF is a non-profit corporation dedicated to promoting computer users' rights to use, study, copy, modify, and redistribute computer programs. The term “free software” became a widely popular definition for this practice of sharing source code for software. The “free” in free software is “free” as in “freedom” and not price.

The term open source came out of a strategy session in reaction to Netscape's announcement of a source code release for their flagship Navigator web browser in January 1998 (INI, 2007). The Open Source Initiative (OSI) was created as an organization to further the ideas of open source and to certify licenses as being true open source.

Open source does not just mean access to the source code, but must follow certain criteria. Based on the Debian Free Software Guidelines written by Bruce Perens, OSI provides an Open Source Definition which asserts nine criteria for open source software. The three main criteria are:

- The ability to distribute the software freely
- The source code’s availability
- The right to create derived works through modification

In addition, there are six more criteria dealing with licensing issues.[1]

Some draw a distinction between the terms Free Software and open source, believing that Free Software is political while open source is pragmatic. I will not go deeper into that discussion in this thesis, but will use the term open source or open source software (OSS) in a wide context throughout this thesis. Other terms have also come up that means more or less the same, like for instance Free/Libre/Open-Source Software (FLOSS), Software Libre and Free and Open Source Software (FOSS).

1 See http://www.opensource.org/docs/definition.html for the whole list

“Open source” is not a precise term with one meaning, but projects claiming to be open source have something in common (Gacek and Arief, 2004). Gacek and Arief (2004) investigated 80 open source projects and found two characteristics that existed in all of them: They adhere to the Open Source Definition, and developers are always users. They found even more characteristics that might vary from project to project. These characteristics included project starting points, motivation among the participants, the community, software development support, licensing, and size. This is not a full list, and even more characteristics might exist.

There are thousands of open source projects ranging from small tools and utilities to database system like MySQL and operating systems like Linux. The Apache web server is the most popular web server in use, and as of January 2007 Apache served 60% of all websites.[2] SourceForge.net, the world's largest open source software development web site, holds over 140.000 projects and has close to 1,5 million users (as of January 2007). So, even though you are not an open source developer or seeking open source software, you are likely using or taking benefit from what open source has to offer. Even large companies writing proprietary software help out in the open source software communities when they can reap long term benefits (DiBona, 2005).

2.1.1 Motivation

So why do they do it? Why do thousands of people devote considerable resources of time and intellect developing a software for free? Although much research and many surveys has focused on what motivates people to engage in open source projects, the answer is still complex and the the reasons numerous.

Hars and Ou (2001) distinguish between motivations which are rooted in the psychology of the individual (internal factors) and motivation which originate from the environment (external factors). The internal factors involves intrinsic motivation, programmers being motivated by the feeling of competence, satisfaction and fulfillment that arises from writing programs, altruism, increasing the welfare of others, and community identification where people identify themselves as a part of the community and align their goals with those of the community. The external factors are future rewards and personal needs.

Bonaccorsi and Rossi (2003) see the production of open source software, first of all, as a form of intellectual gratification. Secondly they see it as an art form. And thirdly and finally, they believe programmers sees the pleasure of creativity. They also states that altruism does not explain the behavior of the open source developers, but at most explains the behavior of people writing software in their spare time.

Findings from the survey of Ghosh et al. (2002) on open source project shows that the most important reasons for people joining an open source project is to learn and develop new skills. Other important reasons they found includes sharing knowledge and skills with other software developers, participating in new forms of cooperation, improving software products or simply participating in the open source scene.

These findings are largely congruent with the findings from The Open Source Technology Group's survey on developers participating in projects at SourceForge. In this survey personal learning and intellectual stimulation from programming was rated highest (Lakhani and Wolf, 2001).

2 http://news.netcraft.com/archives/2007/01/05/january_2007_web_server_survey.html

Goldman and Gabriel (2005) has a long list of explanations as to why people volunteer to do something they can be paid to do, including:

- Need for the product
- Enjoyment, fun, and desire to create and improve
- Reputation and status
- Affiliation
- Identity
- Values and ideology
- Training, learning, reputation outside the community, and career concerns
- Fairness
- Hope of making things better
- Feedback

The reasons for contributing are many, and they even change over time, but no matter if people are doing it for their own personal benefit, for some ideological reason or for the welfare of some community, they are all doing work which we all freely can take advantage of.

2.2 Technical infrastructure in open source

Most open source projects offer at least a minimum, standard set of tools including mailing lists, website, version control, bug tracking and real time chat (Goldman and Gabriel, 2005). Since OSS projects are traditionally open to all, they most often use tools that are open source and available to everyone as well (Erenkrantz and Taylor, 2003). Due to the variety of platform preferences between participants, the tools should also be cross-platform (ibid). Each project uses the tools and the processes that best fit their needs and preferences, but in some areas, a few tools or a single tool is predominant in the market (ibid). When it comes to source control systems, almost all OSS projects use CVS or SVN, and there are two mailing list systems that are commonly used. In other areas, there is no single tool that dominates. Since different tools are used, one can not expect all new developers to have special training in all of them, and to cope with this, the projects should provide clear documentation on techniques that will help unfamiliar developers (ibid).

Almost all OSS projects use a distributed development process with developers located in different places all over the world, and can therefore not rely on face-to-face meetings. This places a strain on the mechanisms used to communicate, and they have to make use of synchronous and asynchronous technologies that can communicate over distance (Holz et al., 1998). The projects primarily rely on mailing lists for almost all communication activities (Cubranic and Booth, 1999). Mailing lists and some of the other tools mentioned will be covered in more detail in this chapter.

2.2.1 Website

Every OSS project needs a website where potential users and developers can find information about the project. When people first hear about a project, the project's website is the first place they will go to find information about it (Goldman and Gabriel, 2005). The website should be the portal to all aspects on the project and the main function should be to present a clear and welcoming overview of the project, and to bind together the other tools (the version control system, bug tracker, etc.) (Fogel, 2005). The site should contain a download page where the latest version of the source code and the program is available. Other topics that websites typically cover are news about the project, user guides, tutorials, archives of mailing lists, and other documentation. Information about how to get involved, sign up for mailing lists and information about the key developers and how to contact them are usually presented as well. Large projects will also have web pages for developer documentation, a road map, descriptions of each module, a list of FAQs about the project and so on. The front page on the website must make it unambiguously clear that the project is open source and which free license the software is distributed under (Fogel, 2005). By not mentioning these topics, the project will lose many potential users.

The website is essential for both new and established developers as well as for users. They will all use the website as a place to meet and a place to find out about the current status of the project. The information presented and how it is organized can help your project to be more successful. A survey mentioned by Goldman and Gabriel (2005, in the chapter: A Community Website) shows that over half the people who responded to the survey did not read any of the mailing lists, but instead relied solely on the website for news. This makes it clear that web pages needs to be up-to-date.

A good website helps create a sense of community and should welcome new participants to this community (Goldman and Gabriel, 2005). To do so, the web pages should include a page listing the major contributors, and it is even better if pictures of the participants are presented together with the name.

2.2.2 Documentation

Documentation is essential (Fogel, 2005, p. 25). Good documentation allows people to use, and equally important in open source projects, understand and modify the software. But even incomplete, rudimentary documentation is better than nothing at all (ibid). Open source communities are generally driven by project members doing what they want to do, and because few programmers enjoy writing documentation, many open source projects are poorly documented compared to proprietary projects. This does not mean that documentation is any less important in open source projects. To align with the read and show that the project members are aware of the deficiencies of the documentation, Fogel (2005, p. 26) suggest to label the areas where documentation is incomplete.

Even though few programmers enjoy writing documentation, Drummond (2000) states that the idea that programmers are poor writers is an unfortunate stereotype. He lists several of the very best hackers, among them Eric Raymond, Richard Stallman and Larry Wall, who are also excellent writers and have written numerous essays, manuals, and technical books.

There are several different types of documentation, including (Wikipedia, 2007a):

- Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components
- Technical - Documentation of code, algorithms, interfaces, and APIs

- End User - Manuals for the end user, system administrators and support staff
- Marketing - Product briefs and promotional collateral

Technical developer documentation is written to help programmers understand the code (Fogel, 2005), and an open source project needs to have good internal documentation for developers (Goldman and Gabriel, 2005). It should be as easy as possible for new developers to get an overview of the software and learn their way around the source code (ibid). The easier it is to learn how to get started, the more developers will be attracted to the project. If the internal documentation is poor or non-existent, the developers are forced to rely solely on the source code. This is a time-consuming and error-prone process and many developers will become frustrated and give up (ibid).

A survey done by Lethbridge et al. (2003) among software engineers shows that documentation is important when learning a new software system. 61 percent rated the available software documentation effective or extremely effective when learning a new software system, and 54 percent reported the same when working with a new software system. This survey was conducted among software engineers, but it is likely that the result would be similar if only open source developers were asked.

The most important documentation for end users is the basics: how to quickly set up the software, an overview of how it works etc. Even though this is the kind of information the writer of the documentation knows all too well, it can be difficult for them to see things from the reader's point of view and they might view some information as too obvious to be worth mentioning (Fogel, 2005).

In addition to the four types of documentation listed above, there are three forms in which open source programs are usually documented (Drummond 2000):

- README files that are distributed with each individual program
- Manual pages or technical references which are also distributed with each program
- HOWTO documents, which are instructional in nature, and usually task- (as opposed to program-) oriented.

Mailing lists are the primary communication channel, making the resulting mailing archives also a source of documentation (Madsen and Nürnberg, 2005).

The causes of poor and lacking software documentation are not unique to open source projects, but in traditional software engineering contexts it is possible and normal to employ technical writers who have dedicated time to write documentation (Yeates, 2006).

The generic challenges to software documentation include skills, time, change, libraries and level, while the issues that aggravate the problem in OS project include (Yeates, 2006):

- Focus on developers – OS projects revolve around developers, pushing other contributors away. See chapter 2.2.2.1
- Excitement – Writing documentation is not perceived as exciting and in OS projects where the contributors have freedom to chose what to do, few contribute to anything that is not exciting.

- Diffuse information – the documentation is usually spread around in mailing lists, forums, chat logs and wiki pages, and few projects have mechanisms for integrating the information into formal documentation.

Spinuzzi (2002) point out accuracy to be one difficulties that can arise from an open system documentation process. When several participants are contributing to the documentation, it can be hard to confirm the accuracy of what they write.

Eric Shepard held a presentation at the Free Software and Open Source Symposium in 2006 called "Documentation in the Open Source World". In this presentation he listed five important C's of documentation[3]:

- Completeness – all topics should be covered and the documentation should be as thorough as possible, but not too detailed.
- Correctness – the given documentation needs to be correct.
- Clarity – the documentation should be written in easy-to-understand language designed for readability. The format should be clear as well.
- Convenience – the documentation should be organized so it is easy to find what you are looking for.
- Consistency – There should be consistency in language, spelling, grammar, colours and formatting.

There is no magic solution to problems with documentation, and workarounds are hard to come by (Yeates, 2006). If the project wants documentation, someone just needs to sit down and write it (Fogel, 2005). The documentation issues can be overcome by consciously and explicitly valuing documentation and the work of writing it (Yeates, 2006). Some of the ways to do this include (ibid):

- Requiring structured documentation along with every contribution of source code.
- Making mailing lists, chat logs, bug reports and other project information accessible to search engines.
- Encouraging new users to contribute documentation as their first contribution to the project. New users are ideal for writing documentation aimed at new users since they have the same point of view.
- Allocation explicit resources to documentation writing.

2.2.2.1 Focusing on the developer

One problem in open source project is that they tend to focus on the code and the developers writing code, and do not pay enough attention to other participants. As Goldman and Gabriel (2005) put it "*There is a tendency in open-source projects to focus on the code, with the result that anyone who is not a developer is often treated as a second-class citizen*". None-developer can have a lot of good ideas and can do other kind of work, like writing documentation and tutorial and they should be encouraged to do so (ibid). People willing to write, organize and keep web pages, servers or documentation up to date, should be blessed and not treated any worse than developers writing code (Goldman and Gabriel, 2005).

3 The presentation is available as a downloadable file: http://cs.senecac.on.ca/fsoss/2006/recordings/

2.2.2.2 Keeping documentation up-to-date

A challenge with project documentation is its degree of freshness. Software changes all the time, leading to out-of-date documentation for most software systems. It is also a problem of keeping end user documentation synchronized with the current version of the software (Erenkrantz and Taylor, 2003). Developers are often hesitant to write user documentation, so when they make a visible change to the software, they may not update the relevant documentation (ibid). Fortunately, Forward and Lethbridge's (2002) survey concludes that document content can be relevant even if it is not up to date. However, they still think keeping the documentation up to date is a good objective.

2.2.2.3 Availability of documentation

Documentation should be available both on the website and in the downloadable distribution of the software (Fogel, 2005). The reason for having it in two places is that people often want to read the documentation before they download the software, but at the same time, the download should supply everything that is needed to use the package. People often want to search for a specific word, and the online documentation should therefore include a link that brings up the entire documentation i one HTML-page (Fogel, 2005). If the document is divided into several chapters, people have to know in which chapter they should look for the information, and this might not be obvious.

2.2.2.4 Documentation technologies

There are several different types of technologies used to write documentation, including word and text processors like MS Word, OpenOffice Writer and Emacs and automated documentation tools like Javadoc or Rational Rose. Word and text processors are flexible and easy to use, but not the most efficient technologies with regards to communication (Forward and Lethbridge, 2002, p. 28). Documentation is an important tool for communication and technologies should enable quick and efficient of communicating ideas (ibid).

Findings from the survey of Forward and Lethbridge (2002), which is based on the most frequently cited technologies among 41 participants, shows that word processors are the most used documentation technology (Table 1: Useful Documentation Technologies).

Documentation Technology	**Frequency**	**Percentage of Participants**
MS Word and other word processors	22	54
Javadoc and similar tools (Doxygen, Doc++)	21	51
Text Editors	9	22
Rational Rose	5	12
Together (Control Centre, IDE)	3	7

Table 1: Useful Documentation Technologies

2.2.2.5 Documentation tools

There have been several attempts to make tools to introduce structure and support the writing of documentation. The basic approach has been to develop hypertext system that models references between documentation and source code (Madsen and Nürnberg, 2005). Examples of this kind of tools include Javadoc (described later), Doxygen[4] and ROBODoc[5].

Madsen and Nürnberg made a prototype tool called Calliope to facilitate developers in aligning their efforts in a common direction at a high level of abstraction. Work related to the Calliope project is also described in Madsen and Nürnberg, among them a tool developed by Cubranic and Murphy called Hipicat. Hipicat applies search algorithms to make the data that is already available, such as CVS logs, mailing archives, IRC chats etcetera, more accessible.

Other documentation tools and approaches have been put forward as well, but the problem has been that open source developers refrain from using these types of tools (Madsen and Nürnberg, 2005). Javadoc is a similar documentation tools and one of the few tools that has won a relatively wide acceptance (ibid). Javadoc is described in chapter 2.2.2.7.

2.2.2.6 Wiki

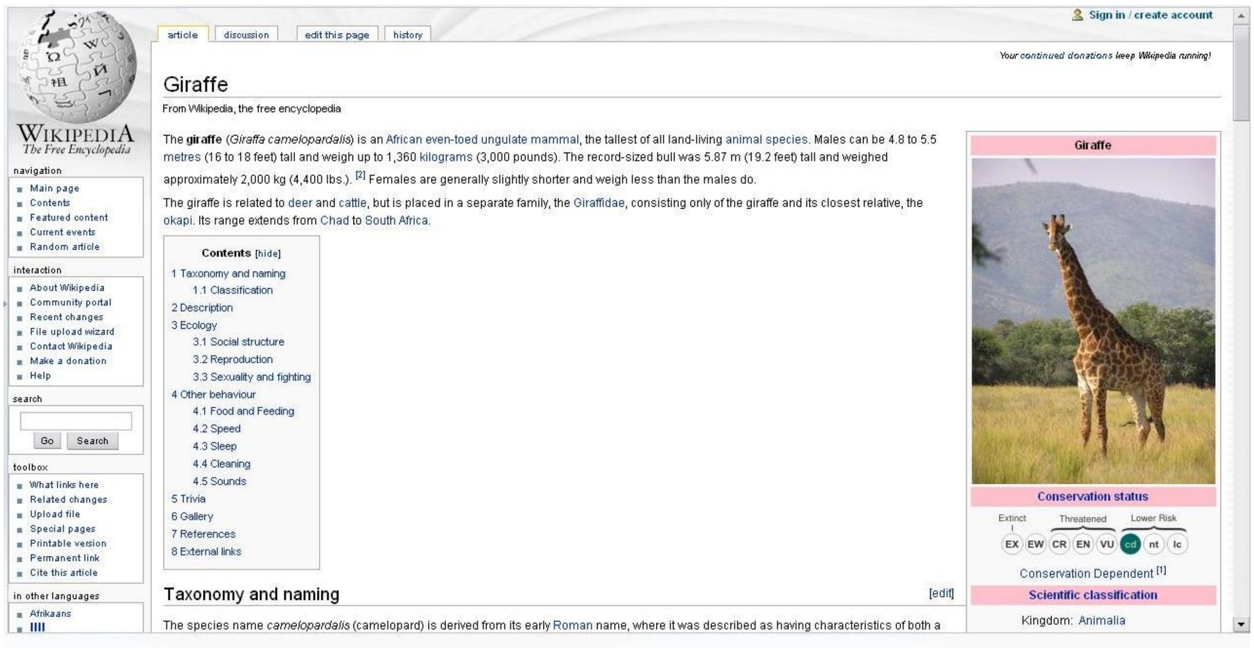
As stated in chapter 2.2.1, a website is very important for an OSS project. A wiki is a kind of website where anyone with a given authority can add, remove, edit or change the content in their own web browser. Some wiki pages allow everyone to change the content, typically without the need for registration, while others are more restricted and only allow a few trusted people to make changes.

The ease of interaction and operation makes a wiki an effective and powerful tool for mass collaborative authoring, either in closed work groups or for the general public on the open Internet (Aronsson, 2002). Wikis are not yet standard tools in open source projects, but they probably will be soon (Fogel, 2005).

As chapter 2.2.2 stated, one critical aspect of software development is documentation, and not only a user manual when the system is ready, but also technical specifications for use by the developers during the project. Traditionally, this has been archived by storing text documents on a shared file server (Aronsson, 2002). This has several drawbacks, including; revision control and the ability to trace a document's history might not be an integrated part of the system, the process for updating and approving a new version of a document can be slow, hypertext links might not be supported and so on (ibid). A documentation system needs to be fast, powerful, easy to use, and highly automated, otherwise developers will avoid using it. This is where a wiki comes in handy.

4 Doxygen homepage: http://www.stack.nl/~dimitri/doxygen/

5 ROBODoc homepage: http://www.xs4all.nl/~rfsber/Robo/robodoc.html



Sign in / create account

article | discussion | edit this page | history

Your continued donations keep Wikipedia running!

navigation
- Main page
- Contents
- Featured content
- Current events
- Random article

interaction
- About Wikipedia
- Community portal
- Recent changes
- File upload wizard
- Contact Wikipedia
- Make a donation
- Help

search

Go | Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Cite this article

in other languages
- Afrikaans
- IIII

Giraffe

From Wikipedia, the free encyclopedia

The **giraffe** (*Giraffa camelopardalis*) is an African even-toed ungulate mammal, the tallest of all land-living animal species. Males can be 4.8 to 5.5 metres (16 to 18 feet) tall and weigh up to 1,360 kilograms (3,000 pounds). The record-sized bull was 5.87 m (19.2 feet) tall and weighed approximately 2,000 kg (4,400 lbs.). [2] Females are generally slightly shorter and weigh less than the males do.

The giraffe is related to deer and cattle, but is placed in a separate family, the Giraffidae, consisting only of the giraffe and its closest relative, the okapi. Its range extends from Chad to South Africa.

Contents [hide]

1 Taxonomy and naming
  1.1 Classification
2 Description
3 Ecology
  3.1 Social structure
  3.2 Reproduction
  3.3 Sexuality and fighting
4 Other behaviour
  4.1 Food and Feeding
  4.2 Speed
  4.3 Sleep
  4.4 Cleaning
  4.5 Sounds
5 Trivia
6 Gallery
7 References
8 External links

Taxonomy and naming [edit]

The species name *camelopardalis* (camelopard) is derived from its early Roman name, where it was described as having characteristics of both a

Giraffe

Conservation status

Extinct | Threatened | Lower Risk

EX EW CR EN VU cd nt lc

Conservation Dependent [1]

Scientific classification

Kingdom: Animalia

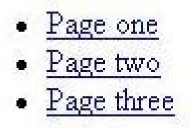
Figure 1: A screenshot from Wikipedia, one of the best-known wikis.

Wiki pages are written in a special simplified markup language, sometimes known as wikitext. This markup language is an attempt to simplify the syntax usually used to write web pages, called HTML– Hypertext Markup Language.

The syntax to write a bulleted list with links in HTML is:

```
<ul>
  <li><a href="page1.html">Page one</a></li>
  <li><a href="page2.html">Page two</a></li>
  <li><a href="page3.html">Page three</a></li>
</ul>
```

In a web browser this would look like this:



- Page one
- Page two
- Page three

Figure 2: The bulleted list rendered in a web browser.

The style and syntax varies between different wiki software, but to write the same bulleted list with the syntax of MediaWiki, one wiki software, it would look like this:

```
* [[Page1|Page one]]
* [[Page2|Page two]]
* [[Page3|Page three]]
```

HTML, which is many cryptic tags, is not especially human-readable. The idea behind the wiki syntax is to lower the barriers use so non-technical can easily contribute without having to learn these cryptic tags.

The advantages of using a wiki include (Stafford and Webb, 2006):

- Good for writing down quick ideas or longer ones, giving you more time for formal writing and editing.
- Instantly collaborative without emailing documents, keeping the group in sync.
- Accessible from anywhere with a web connection (if you do not mind writing in web-browser text forms).
- Serves as an archive, because every page revision is kept.
- Exciting, immediate, and empowering--everyone has a say.

Most people, when they first learn about the wiki concept, assume that a website that can be edited by anybody will suffer from “trolls” writing malicious or wrong information (Aronsson, 2002; Goldman and Gabriel, 2005). This has turned out to be a small problem in most cases, since people can easily see the changes that are done and all pages are kept under version control, making it easy to roll-back to a previous version (Aronsson, 2002).

Wiki pages are becoming more and more common in open source projects, but there are a few of things to look out for when using wikis. Too often they suffer from (Fogel, 2005):

- Lack of navigational principles
- Duplication of information
- Inconsistent target audience

The common solution to all these problems is to have editorial standards and demonstrate them by editing pages to adhere to them (Fogel, 2005).

Other disadvantages include that is it not obvious how to set up or back up wiki software, the user needs to learn and understand the concept of text markup used in the wiki and the wiki generally tends to get disorganized and chaotic(Stafford and Webb, 2006). A wiki is not an administrative panacea and a certain amount of maintenance and standards is needed to avoid a disorganized wiki (ibid).

People who are unfamiliar with wikis can also see it as a barrier to contribute. They can be afraid of what will happen to the information they write, and can also be unsure about what type of contributions are acceptable (Goldman and Gabriel, 2005).

2.2.2.7 Javadoc

Javadoc is a computer software tools for generating API documentation into HTML format from Java source code. A Javadoc comment is a specially marked comment in the source code that describes the code. The comment begins with /** and ends with */. For most Java class libraries, the Javadoc is the only documentation (Goetz, 2002). Javadoc is a great reference tool, but it is not a great tool for learning how Java classes or methods are organized and how they should be used (ibid).

Most Java classes do not have Javadoc, and when they do, the Javadoc often contain only the most basic information about what a method does. Effective Javadoc should at least include descriptions of (ibid):

- How classes relate to each other

- How methods affect the state of the object
- How methods communicate error conditions to their callers and what errors they might signal
- How the class deals with being used in a multithreaded application
- The domain of methods' arguments and the range of their return values

A positive side effect of writing good Javadoc is that it becomes a sort of code review where the architecture of a class or method, and how they relate to each other is explored (Goetz, 2002, Writing Javadoc is a form of code review). If a package, class or method is hard to document, then it is probably trying to do more than one thing, and should perhaps be re-engineered (ibid).

2.2.2.8 FAQ

A FAQ ("Frequently Asked Questions" document) is a document that covers questions asked by the participants and answers to these question, and should contain the questions that are actually asked instead of what might be asked. Since it is impossible to know upfront the question people might ask, it is impossible to sit down and write useful FAQs from scratch. The FAQ is often the first place users look to solve a problem and it can be on of the best investment for a project when it comes to educational pay-off (Fogel, 2005).

2.2.3 Mailing lists

Mailing lists are the most used communications form in open source projects; "*[they]... are the bread and butter of project communications*" (Fogel, 2005, p. 37). All OS projects, almost without exception, rely primarily on mailing lists for nearly all communication activities (Cubranic and Booth, 1999). Cubranic and Booth give several reasons for choosing this low-tech approach. First and foremost, e-mail is the lowest common denominator for Internet communication, which makes it easy to get people to participate or even just follow the discussion. Secondly, the distributed nature of open source projects precludes the usage of synchronous communication. Thirdly, and finally, the structure of open source projects is minimal and developers contribute when they have time and feel like doing it. The asynchronous nature of e-mail means that participants can take part in communication at their leisure.

It is important that all discussions about an open source project is done in the open, and mailing lists or newsgroups are common ways of achieving this. These discussions include announcements, bug reporting, problems and how to solve them, design issues, and proposals for future work (Goldman and Gabriel, 2005). A small project may need only a single mailing list, but to manage these different kinds of discussions in a large, active project, several mailing lists can be necessary. By looking at some of the more successful OSS projects, like the Apache web server and Maven, you will see that there are often a number of different mailing lists in OSS projects; the most common being (Nordal, 2006):

- Users' list for interaction between and among end-users and developers.
- Developers' list for interaction between internal and external developers.
- Issue list for mails announcing activities in the issue tracker.

- Commit list for announcing activities in the source code repository.

The point is not to have many mailing lists. A mailing list should be alive with activities, and in general, it is better to have too few mailing lists than to have too many (Goldman and Gabriel, 2005). When the traffic on one mailing list gets too intense or people start discussing different topics over a long period of time, then a new mailing list should be established. Large OSS projects with developers in many countries may have different mailing lists in different languages (ibid).

It is important to keep an archive of each list and make searching them easy (ibid). This is useful for new developers and new users so they can see if a particular issue has already been discussed. It is also a nice way to keep a group record.

2.2.4 Public code archive

A prime requirement for an open source project is that the source code is publicly available (Goldman and Gabriel, 2005), and it should be possible to get the latest version of the source code at any time (Fogel, 2005; Goldman and Gabriel, 2005). The way to achieve this is to use a version control system (Fogel, 2005).

Most projects will adopt some sort of source control management (SCM) system, and the most widely used source control system in open source project is Concurrent Versioning System, CVS (Erenkrantz and Taylor, 2003; Goldman and Gabriel, 2005). There has been a recent trend in seeking tools that can replace CVS (Erenkrantz and Taylor, 2003) and Subversion, often referred to as SVN, is one attempt at that. SVN is meant to be a better CVS and a compelling replacement for it in the open source community.[6]

Version control helps with virtually every aspect of running a project, from communication between developers, code stability and release and bug management, to experimental development efforts and attribution and authorization of changes by particular developers (Fogel, 2005). A version control system manages files and directories over time in a central repository where the repository is much like other file servers, except that it remembers every change that have ever been done to the files and directories. This makes it possible for multiple developers to work independently while allowing them to remain updated and synchronized with the rest of the team (Erenkrantz and Taylor, 2003). Since the history of every file is recorded, is it possible to examine the history of the repository or recover an old version of data if problems occur (Collins-Sussmann et al., 2006).

Typically in open source projects, version control systems allow anyone to read and copy the source code, but only authenticated developers are allowed to update the source code in the repository.

2.2.5 Issue tracker

An issue tracker is used to keep a record of known bugs and other issues, and is a must in OSS projects (Goldman and Gabriel, 2005). The tracker goes by several names, such as bug tracker and issue tracker, since these tools are usually fit for tracking any kind of issues (bugs, tasks, request, ideas, etcetera). There are different

6 See the Subversion homepage for more information: http://subversion.tigris.org/

types of bug tracking tools, including web-based bug databases and tools that can be used via e-mail.

The issues that are registered in a issue tracker have various kinds of tags or attributes connected to them. These attributes can be status (e.g. new, assigned, resolved, reopened, closed), priority (blocker, critical, high, normal, trivial), type (task, new feature, defect, enhancement). In addition, issues can be assigned to a specific release and to a particular developer who are responsible for resolving them.

The classic issue life cycle looks like this (Fogel, 2005, p. 54-55):

1. Someone files an issue and provides a summary and an initial description.
2. Others read the issue and make comments about it.
3. The bug gets reproduced to confirm that it is a real bug.
4. The bug gets diagnosed; its cause is identified, and if possible, the effort required to fix it estimated.
5. The issue is scheduled for resolution.
6. The bug gets fixed.

There are other possible life cycles where the issue gets closed because it is not a bug, the issue gets closed because it is a duplicate or other small variations (ibid).

Issue trackers are usually open to everyone and anyone may file an issue, look at an issue or browse the current issues. For many people an open issue tracker is one of the strongest signs that a project should be taken seriously (Fogel, 2005). Since both users and developers can file issues, and users tends to be a prime source of bug reports, the process of reporting bugs should be easy. As Goldman and Gabriel (2005, chapter 6) puts it: "*Keep in mind that they have already suffered by discovering bugs--they may have lost their work and undoubtedly lost time--so don't make it painful for them to submit bug reports too*". One way to solve this is to have different ways for users and developers to report bugs. The users report the bugs they discovered, and the developers, with more information and insight to the project or software, can file a more informative issue.

Fogel (2005) argue that it is important to have the tracker connected to a mailing list, so that every change to an issue causes a mail to go out describing what happened. This automatically informs the project members of activities in the tracker and helps encourage and stimulate timely reactions to the registered issues.

2.3 Social infrastructure in open source project

In an open source project, software building and community building are intertwined. As the software matures, the community needs to keep up with it. Developers may be physically and geographically separated, but a good community can make them feel like they are working together in the same room (Fogel, 2005). The more they feel a part of the community, the more time they will spend on the project. To enhance the feeling of community, everyone involved with the project should know what is happening with it (Goldman and Gabriel, 2005). Distributed software development also places a strain on the communication mechanisms used in the project since developers are not co-located (Erenkrantz and Taylor, 2003).

The following sections will introduce common practices and ways of conducting software development in open source projects.

2.3.1 Leadership

The traditional approach to managing a large group of workers has been to establish a strict hierarchy of managers controlling the activities of the people below them (Goldman and Gabriel, 2005). This is time consuming, inefficient and requires a lot of managers to manage the workers (ibid). Open source projects, on the other hand, are self-organized and work towards shared goals where the actual people using and developing the software discuss what needs to be done on mailing lists and newsgroups. The unconstrained nature of the open source process might seem to leave little scope for a leadership, but this is incorrect. Most successful open source projects display a clear hierarchical organization (Bonaccorsi and Rossi, 2003) and by looking at successful OSS projects like the Apache Software Foundation and SourceForge.net, we can find examples of strong leadership and management (Nordal, 2006).

The governance structures of open source projects vary a lot, but the leaders share some common features. Mostly, the leader is the person who started the project by developing the initial code for the project or making another important contribution early in the project's development. The initial experience is important in establishing the credibility needed to manage the project. Leadership in OSS projects is not about being in charge, making decisions, or give orders, but about having a vision and work with others to make it happen (Goldman and Gabriel, 2005).

Fogel (2005) identifies two different leadership styles most commonly found in OSS projects: The benevolent dictator (BD) and consensus-based democracy. These two styles are the idealized extremes and most projects can be placed somewhere in a continuum between them. In the BD model, final decision-making authority rests with one person, but generally, the benevolent dictator act more like a judge and does not make all the decisions personally. The BD leader will normally let things work themselves out through discussions and experimentation, and only intervene when considered necessary.

Another model commonly used by open source projects is the meritocracy model (Erenkrantz and Taylor, 2003). In this model, all members share power equally and there is no direct leader of the project. People gain power by sustained contributions over time and those who have demonstrated their competency through their work on the project, are the ones who make the decisions (Goldman and Gabriel, 2005).

Lerner and Tirole (2002, p. 21) list four tasks a leader must do:

- Provide a vision
- Make sure that the overall project is divided into smaller and well-defined tasks (modules) that individuals can tackle independently from other tasks
- Attract other programmers
- Keep the project together (prevent it from forking or being abandoned)

Edwards (2000) disagrees with several of these tasks, and argues that defining modules and tasks in an open source software development project is not the task of the leader, but the leader should encourage the creation of modules. He also claims that OSS projects gain users and co-developers from those searching to solve a problem, and not because of the leader or his or hers actions. The project leader can of course, according to Edwards, increase the probability of people finding the project by promoting it on relevant search engines and web pages.

One determinant of project success appears to be the nature of its leadership (Lerner and Tirole, 2002). Max Weber in Lerner and Tirole (2002) gives some attributes which underlie successful leadership. The first attribute is that the programmers must trust the leadership. The programmers have to believe that the leader's objectives are sufficiently congruent with theirs and not polluted by ego-driven, commercial, or political biases. Secondly, the leader must clearly communicate his/her goals and evaluation procedures. Edwards (2000) claims that the properties associated with good leadership are difficult to apply and find in OSS projects. He even suggests that the term “leader” should be abandoned in OSS development projects, and the term “maintainer” should be used instead to describe the key person in a given project.

2.3.2 Coordination

Project coordination can be defined as the attempt to get the right information to the right people at the right time (Holz et al., 1998).

Since open source development is a collaborating process between participants dispersed worldwide it calls for other types of coordination than proprietary development where all participants are co-located. Open source projects cannot rely on face-to-face meetings, but have to make use of other forms of technology to coordinate the project and make decisions over distance (Cubranic and Booth 1999). Mailing lists are one common technology used to archive this kind of coordination over distance.

OSS projects are missing many of the traditional mechanisms used to coordinate software development, such as plans, system-level design and scheduled and defined processes (Mockus et al., 2000).

2.3.3 Decision making

All decision making in an open source project should happen either on the project's public mailing lists or in a public community meeting (Goldman and Gabriel, 2005). The disadvantages of public list discussions included the delay of using e-mail for conversations, the hassle of volunteers who think they understand all the issues, when they actually do not, rude or insulting behaviour because people will say things in e-mail that they would never say face-to-face, and so on (Fogel, 2005). Public discussion also usually takes more time to reach a conclusion then proprietary development groups, but even though public discussion may be slow, they are always preferable in the long run (ibid). Few volunteers will stick around in a project where a secret group makes all the big decisions (ibid).

The process of making a decision varies from one open source project to the next, but it is often based on the idea of a meritocracy (Goldman and Gabriel, 2005). Following this idea, the originator of the code, or the model owner, often has the final say, but it only works if the benevolent dictator can maintain the respect of the developer community (ibid). If this respect is lost, the community will call for a replacement.

2.3.4 Releasing and distributing

Every time someone checks in a change to the source code repository, that is a new release (Goldman and Gabriel, 2005). This means that active developers are guaranteed to be working on the most recent code and do not have to spend time trying to fix a bug somebody already has fixed. In addition, the developers contributions can be used and given feedback on immediately. Users, on the other hand, might want more stability in the software they rely on. To satisfy these two conflicting needs, many OSS projects do a series of frequent, small, incremental releases. “Release early and release often” is a strong community norm in OSS development (Raymond, 1998).

Goldman and Gabriel (2005) argue that the release process for an OS project is very similar to that used for proprietary products, except that OS projects tend to be more loosely organised (Goldman and Gabriel 2005, chapter 6. How To Do Open-Source Development), while Fogel think there is a difference (Fogel 2005, p. 111). Fogel (2005) argues that a corporation can ask all developers to put everything on hold and fully focus on an upcoming release, while in an OSS project not everyone will be interested in helping out with an ongoing release. Volunteers contribute to the OSS project for all sorts of reasons, and even though they are not interested in helping with an upcoming release, they might still want to continue regular development work while the release is in process. As a consequence, the release process tends to take longer time in OSS project, but it is less disruptive compared to commercial release processes.

Code freezing is not a good idea in OSS project, because ongoing development is likely to continue during the release process. Developers who want to continue their work on new and experimental modules that will not be included in the release, might abandon the project if they cannot check in their code and test it promptly because of code freezing. The solution to this problem is to use a release branch (Fogel, 2005; Goldman and Gabriel, 2005). A release branch is just a branch in the version control system where the release activity can proceed, while normal development continues in the main trunk.

Before making a release, it must be decided which changes will be in the release, and which will not. There are several systems used to do this work of stabilizing a release. Two of the most popular systems are the dictatorship model with a release owner and a more democratic vote system (Fogel, 2005). In the dictatorship model, the group agrees on one person to be the release owner. There is a discussion about what makes it into the release, but the release owner has the authority to make final decisions. With a voting system, the majority makes the decisions. Not every participant in the project is necessarily given a vote, and having a voting system raises the question about who gets to vote. There are several ways of solving this, but one approach is to use the voting system itself to choose new voters.

A release manager can be used to coordinate the release process. The release manager is quite different from the release owner, and the manager's job includes helping to keep track of what goes into the release and what is not yet ready, recruiting testers and coordinating the testing process (Fogel, 2005 and Goldman and Gabriel, 2005).

When most of the known bugs have been fixed and the release is becoming stable, the release should be tested and approved by developers (Fogel, 2005). Raymond (1998) introduces what he calls “Linus' Law”: Given enough eyeballs, all bugs are shallow. He argues that more users find more bugs because adding more users adds more different ways of stressing the program. "*Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone"* (Raymond, 1998, The eight lesson). Goldman and Gabriel (2005) follow up on this and suggest that a beta release should be put out before the official release. They argue that more people are willing to try out a beta version which has already undergone substantial testing, and that these people, the second batch of testers, will help catch the remaining bugs and improve the quality of the release.

Once all major bugs are fixed and the release is approved, it should be packed for distribution and announced to the world. The release should be placed into the project's download area and made available to users as both source code packages and binary packages in different file types.

It is important to give each release a unique release number so that everyone knows what is the newest release is and that bugs can be reported to the right version of the source code (Fogel, 2005; Goldman and Gabriel, 2005). The most common release number system is the three-component system where the first component is the major number, the second is the minor number and the third is the micro number. A project usually has guidelines to describe what kind of changes are micro, minor and major. There are many different methods and conventions for how many components to use, and what they mean, but the differences tend to be minor (Fogel, 2005).

2.3.5 Guidelines

Developer guidelines are social guidelines which explains how the developers interact with each other and with the users, and how things get done. When some one is considering contributing to the project, the first thing they will look for are the guidelines (Fogel, 2005).

Open source projects usually have a high turnover rate, and it is thus important to have developer guidelines which allow new developers to familiarise themselves with the processes and tools being used in the project. If given good guidelines, new developers can contribute to the project in an intelligent manner (Erenkrantz and Taylor, 2003). Guidelines can also prevent conflicts. The idea is that most conflicts will be resolved peacefully by creating and following pre-made guidelines (ibid).

2.3.6 Communication and knowledge sharing

A common problem in open source projects is to understand what the other participants are currently working on (Erenkrantz and Taylor, 2003). It can even be hard to identify the participants who are currently active in the project (ibid).

Asynchronous mechanisms for communication, like e-mails, makes it easier for more people to participate and are usually preferred. If synchronous methods are used, some participants may not be able to contribute to a discussion because of different time zones, busy time schedules or other appointments (ibid). When synchronous communications are used, and not all developers can participate, it is essential to make some form of archive of the communication (ibid).

Mailing lists are great for discussions, but not everyone has the time to follow them. To keep people informed on current issues, Goldman and Gabriel (2005) suggest having a project newsletter published on a regular basis. This newsletter could contain links to interesting discussions on the mailing lists, articles covering project related news, articles on some individual who is doing interesting work on the project, pointers to press articles about the project, news on related software and so on.

Open source projects with public discussions usually have a larger diversity of viewpoints than proprietary projects, and therefore the conclusion is often of higher quality (Goldman and Gabriel, 2005). This can lead to a shorter overall development cycle because subsequent work will not be discarded because issues were raised after, rather than during, the discussion period (ibid). Fogel (2005) lists other beneficial side effects gain from public discussions:

- The discussion will help train and educate new developers. You never know how many eyes are watching the conversation; even if most people don't participate, many may be tracking silently, gleaning information about the software.
- The discussion will train you in the art of explaining technical issues to people who are not as familiar with the software as you are. This is a skill that requires practice, and you can't get that practice by talking to people who already know what you know.
- The discussion and its conclusions will be available in public archives forever after, enabling future discussions to avoid retracing the same steps.

Small groups can, and should, discuss in private to work up proposals and suggestions, but these proposals should be presented to the community for feedback as soon as possible (ibid). There are also some discussions that must be held in private, but the principle should always be: "*If there's no reason for it to be private, it should be public*" (Fogel, 2005, p. 31).

2.4 Former research on DHIS 2

Several master theses has been written about the DHIS 2 software, most noticeable are the theses written by Nordal (2006) and Øverland (2006). Nordal's thesis explores a lot of the problems arising from establishing the DHIS 2 project as an open source project. He was one of the central participants in the early period and took part in the discussions about the tools, frameworks and programming languages being used in the development. He also contributed greatly to the development of the software and was part of the first initiative to establish a HISP node in Vietnam. Empirical data and findings from his thesis are used as background information about the DHIS 2 software and its development process in this thesis.

Øverland is one of the core developers who participated in the effort to establish a local development team in Vietnam. Øverland supplies more information about the Vietnam node which complements Nordal's work, but this is not used extensively in this thesis.

This thesis is mainly based on experiences and findings from after the time period covered in the two other thesis.

3 Health Information Systems Programme (HISP)

This section will give a brief historical overview of the HISP project and the development of the health information systems DHIS 1 and DHIS 2. The technical and social infrastructure surrounding the DHIS 2 development will also be presented and discussed.

3.1 HISP history

The Health Information Systems Programme, HISP, is an ongoing research and development project which was initiated after the fall of apartheid in South Africa in 1994. The project started as a collaboration between public health, medical and computer science departments at the University of Oslo(UiO) in Norway, the University of Western Cape(UWC) in South Africa and the Ministry of Health in South Africa.

As a legacy of apartheid, South Africa was left with one of the least equitable health care systems in the world where 60% of the resources were being used by the private sector, serving only 20% of the population (Braa and Hedberg, 2002). The new government launched the Reconstruction and Development Program (RDP) to reconstruct and redevelop the communities that suffered under apartheid. One of its goals was to develop a new national health information system. As a part of this, the Strategic Management Team, established by the RDP, proposed a pilot project to develop district health and management information systems. HISP was established in 1996 in three health districts in and around Cape Town to be a part of this pilot project.

Braa et al. (2004, p. 343) define the primary HISP research goal as follows:

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

HISP wanted to empower the health districts and focus on the use of the information at district level. These efforts transformed into two main areas for research and implementation (Braa and Hedberg, 2002):

- Development of Essential Data Sets and standards for primary health care data
- Development of a District Health Information Software (DHIS) supporting the implementation and use of such data sets

An Essential Data Set is defined as a set of the most important data elements, selected from all vertical primary health care programmes, that should be reported by health service providers on a routine basis. The benefits of developing these essential dataset is to clearly define which parameters should be monitored and used (Shaw 2005).

The first essential data set was implemented in 1997 in all local government health facilities in the Cape Metropole (including the HISP pilot districts). It later spread to the whole province of Western Cape. In 1998 HISP released the first implementation of the District Health Information Software (DHIS), supporting the collection of standardized health care data.

In February 1999, the Department of Health in South Africa adopted the strategies, processes and software developed in the pilot districts as the national standard (Braa and Hedberg, 2002). By 2001, HISP was established in all provinces and districts in South Africa. This later evolved into HISP efforts in other countries like Mozambique, India, Vietnam, Tanzania, Ethiopia and Zanzibar.

In the beginning of the 2000s, HISP approached the European Union and formed the BEANISH (Building Europe-Africa Network for applying IST in the Health care sector) network. This initiative seeks to involve various institutional actors (government, universities, private sector and non-governmental organisations) to strengthen and extend an Europe-Africa collaborative network. The focus of the network is to support cooperation, learning and innovation in mutually beneficial ways.

3.2 DHIS history

The free and open source District Health Information Software (DHIS) application is a flexible database tool for capturing data elements and conducting planning and analysis on registered routine data, semi-permanent data, audit data and survey data. DHIS is designed to support health workers and managers at all administrative levels through a balance between flexibility and standardization, and with a strong emphasis on using information for local action.

The basic idea is to give health workers an overview of the situation in a district or region. This overview then serves as a tool in determining where to concentrate extra resources and efforts.

3.2.1 DHIS 1

The first prototype of DHIS 1 was released for pilot testing in March 1998, and went through a series of rapid prototype cycles with new releases on a weekly or even daily basis. The software was at this time developed by a two-person team; one system analyst/designer and one developer. In 2000-2001 additional developers took part in the software development. After that period, the development was done primarily by one developer, who hired extra help when needed. Masters students connected to the HISP network have also participated, but mostly on the implementation side.

The software was developed with the following objectives (Braa and Hedberg, 2002):

- Shift of control of information systems from central towards local levels, i.e. towards more equal control between central and local levels.
- Local flexibility and user orientation – it should be easy to adapt the software to local conditions.
- Support for health sector reform towards decentralization and the development of health districts, i.e. integrating the vertical flows at district level.
- Empowerment of local management, health workers and communities.
- Horizontal flow of information and knowledge, based on the principle of free access to all anonymous, aggregated health data/information.

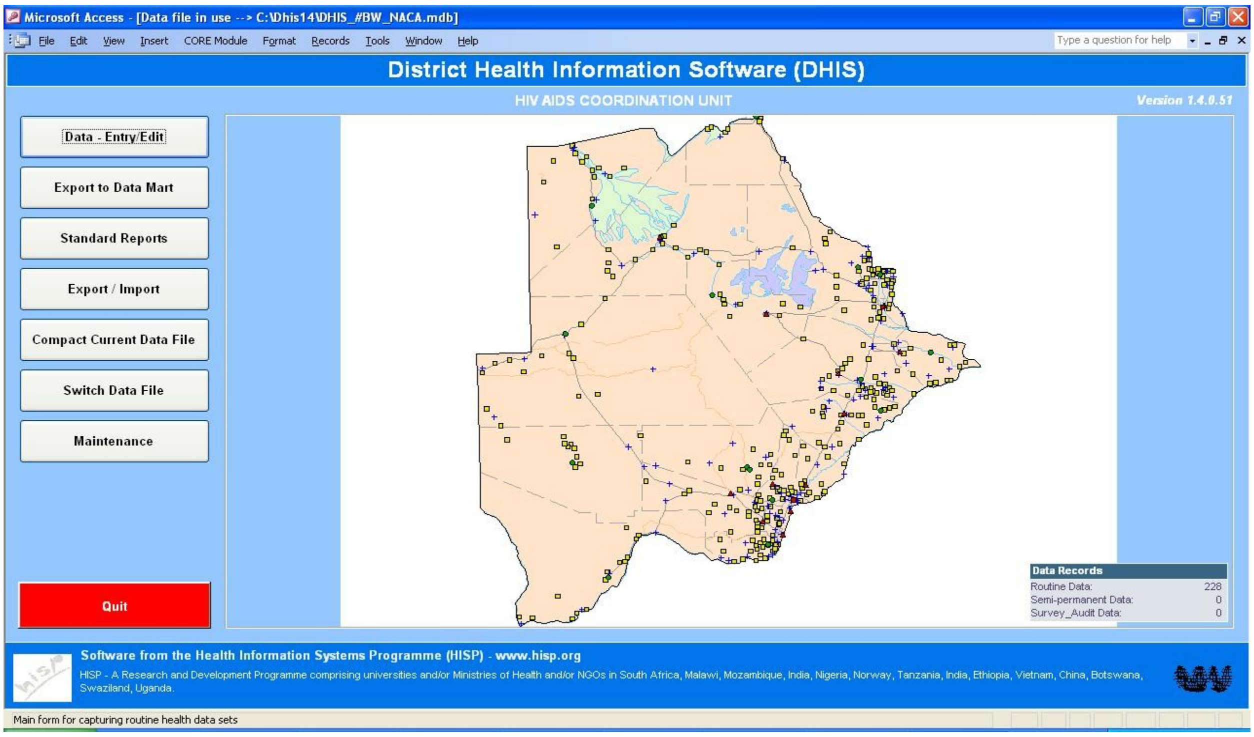

Microsoft Access - [Data file in use --> C:\Dhis14\DHIS_#BW_NACA.mdb]
File Edit View Insert CORE Module Format Records Tools Window Help
Type a question for help
District Health Information Software (DHIS)
HIV AIDS COORDINATION UNIT
Version 1.4.0.51
Data - Entry/Edit
Export to Data Mart
Standard Reports
Export / Import
Compact Current Data File
Switch Data File
Maintenance
Quit
Data Records
Routine Data: 228
Semi-permanent Data: 0
Survey_Audit Data: 0
Software from the Health Information Systems Programme (HISP) - www.hisp.org
HISP - A Research and Development Programme comprising universities and/or Ministries of Health and/or NGOs in South Africa, Malawi, Mozambique, India, Nigeria, Norway, Tanzania, India, Ethiopia, Vietnam, China, Botswana, Swaziland, Uganda.
Main form for capturing routine health data sets

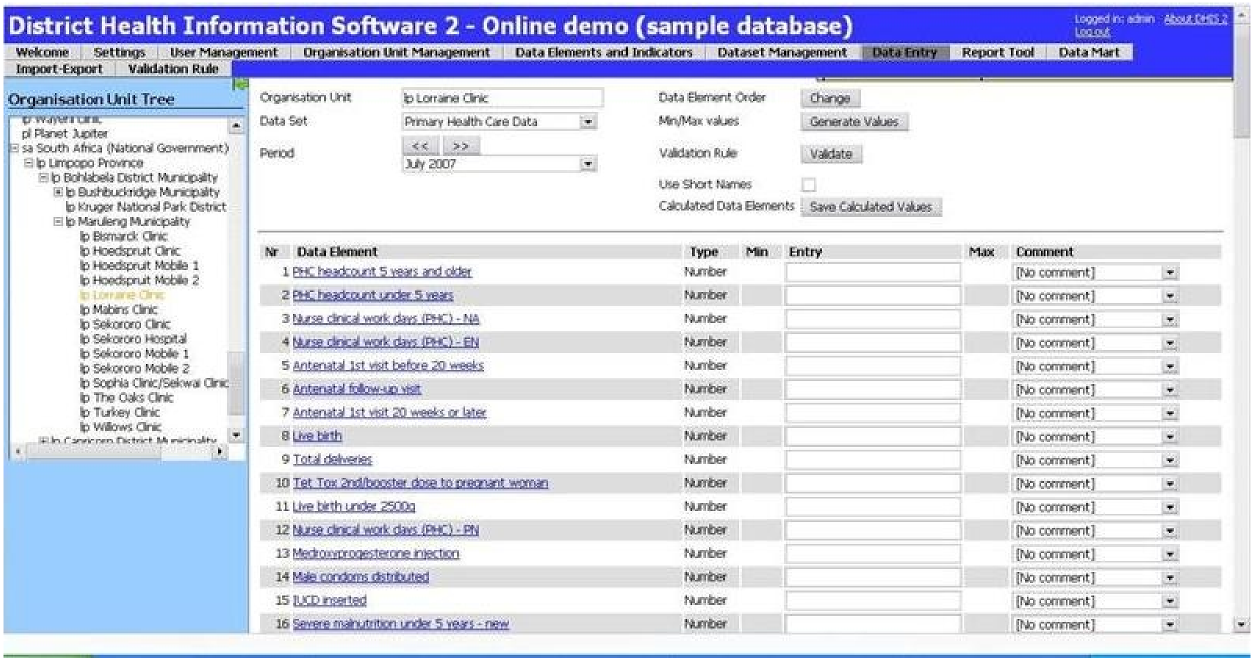
Figure 3: Screenshot of the welcome screen of DHIS 1.4

The software has been developed using Visual Basic (VB) and runs in a Microsoft (MS) environment with Windows and MS Office (Access and Excel). MS Office was selected mainly because it was a standard among potential users already.

Multiple versions of DHIS exists. The DHIS version 1.3 is based on Microsoft (MS) Office. The DHIS version 1.4 also relies on MS Office to some extent, but it is at the moment regarded as the last version to rely predominantly on Office/VBA.

DHIS 1 has been used on a national basis in South Africa since 2001, and because of its success it has spread through several other developing countries.

3.2.2 DHIS 2



pl Planet Jupiter
sa South Africa (National Government)
lp Limpopo Province
lp Bohlabela District Municipality
lp Bushbuckridge Municipality
lp Kruger National Park District
lp Maruleng Municipality
lp Bismarck Clinic
lp Hoedspruit Clinic
lp Hoedspruit Mobile 1
lp Hoedspruit Mobile 2
lp Lorraine Clinic
lp Mabins Clinic
lp Sekororo Clinic
lp Sekororo Hospital
lp Sekororo Mobile 1
lp Sekororo Mobile 2
lp Sophia Clinic/Sekwai Clinic
lp The Oaks Clinic
lp Turkey Clinic
lp Willows Clinic

Organisation Unit: lp Lorraine Clinic
Data Set: Primary Health Care Data
Period: July 2007
Data Element Order
Min/Max values
Validation Rule
Use Short Names
Calculated Data Elements

Nr	Data Element	Type	Min	Entry	Max	Comment
1	PHC headcount 5 years and older	Number				[No comment]
2	PHC headcount under 5 years	Number				[No comment]
3	Nurse clinical work days (PHC) - NA	Number				[No comment]
4	Nurse clinical work days (PHC) - EN	Number				[No comment]
5	Antenatal 1st visit before 20 weeks	Number				[No comment]
6	Antenatal follow-up visit	Number				[No comment]
7	Antenatal 1st visit 20 weeks or later	Number				[No comment]
8	Live birth	Number				[No comment]
9	Total deliveries	Number				[No comment]
10	Tet Tox 2nd/booster dose to pregnant woman	Number				[No comment]
11	Live birth under 2500g	Number				[No comment]
12	Nurse clinical work days (PHC) - PN	Number				[No comment]
13	Medroxyprogesterone injection	Number				[No comment]
14	Male condoms distributed	Number				[No comment]
15	IUCD inserted	Number				[No comment]
16	Severe malnutrition under 5 years - new	Number				[No comment]

Figure 4: Screenshot of data entry in DHIS 2.

HISP has all along defined DHIS as open-source software and the code for DHIS 1 is open and free, but the modules being used from Microsoft are not. This means you have the possibility to change DHIS to suit your needs, but you still have to purchase Microsoft licenses to run the software. Microsoft licenses can be expensive, especially for developing countries, so the DHIS 2 is now being developed using open source Java frameworks and tools: The Spring Framework, Hibernate, WebWork, Maven, and JUnit.

The license for DHIS is written by HISP and has been bundled with and made available through the application. It is in principle quite similar to the Lesser GPL license. In short it says that anyone can use the DHIS software however they like, except for commercial purposes. The license is specific to the DHIS software, and not an official OSS license (defined by the Open Source Initiative as a license conforming to the Open Source Definition).

In 2003, DHIS 1.3 was the current stable version of the software, with a version 1.4 on the way. Evaluations of the software were conducted at this time, and discussion and criticism around the core issues in the software was triggered from a number of central persons tied to the HISP network.

The core of the discussion was as follows (Nordal, 2006, p. 30-31):

- DHIS was not web-enabled, and the technologies that it was built on made it practically impossible to change that. Integration between users (exporting and importing data) was done manually (exporting to file and manually sending it via e-mail even between users with Internet connectivity.
- The software was a standalone desktop application that would only run on MS Windows, using the Access DB. A web-enabling of the software would open it up for use on all platforms, using web browsers as clients.

- Users were experiencing major performance issues with the MS Access database engine because of the amount of data handled by DHIS. Being able to use the DHIS software with a full-blown relational database management system (RDBMS) was starting to become critical for some.
- DHIS was the result of a lengthy prototyping approach, where add-ons have been built on add-ons and so on. DHIS has therefore often been described as an onion, where new layers of functionality have been added to the existing layers, creating an overly complex data model.
- A major issue that concerned many was the lack of a layered architecture. It was argued that splitting the application into a three-tier architecture model and modularizing it would be critical for further development and adoption of DHIS.
- DHIS is built with technologies and written in a language that were starting to become outdated, and some of the participants were arguing that HISP needed to start looking at other alternatives that are more suitable for the continued evolution and development of DHIS

Talk and discussions around the DHIS 2 continued, but it was not until May 2004 that HISP hired a researcher to start working on it full time. Another PhD student joined in June, and together they started the initial phases of the DHIS 2 project, involving a review of potential technologies and frameworks. The previous versions of DHIS acted as a requirements specification, but in addition, two new technical requirements were considered vital:

- The system needed to be platform independent and able to run on most relational database management systems.
- It must be possible to develop both web based and desktop modules for the system

Java was selected as the programming language for several reasons; it has a strong presence at UiO where Java is used in most of the programming courses, it is a popular programming language, it provides services which make it highly suitable for web application development and it is platform independent.

While the development of DHIS 1 started from scratch with little or no experience with HIS, the development of DHIS 2 could benefit from the experience and findings from the work with DHIS 1, resulting in less need for user participation and prototyping. The requirements was to a large extent defined by DHIS 1; the initial focus was to copy and improve the functionality from the existing system with a web based application. While most of the development of DHIS 1 was done by a hired team of professional developers, DHIS 2 is primarily developed by students and researchers.

DHIS 2 has (as of July 2007) reached its seventh milestone release, and has been translated into English, Norwegian, Vietnamese, Hindi, French, Malayalam, Telugu, Amharic, Gujarati and Kannada. The software is implemented and used in several districts in Vietnam and India.

3.3 Development network

HISP consists of different nodes bundled together in the HISP network. Braa et al. (2004) define the nodes as being on two levels, different countries and set of institutions, typically universities, in the various countries. The primary nodes, as defined by Braa et al. (2004), are South Africa, Norway, Mozambique, Tanzania and Ethiopia. Since this article was written, HISP has expanded further, with primary nodes in among other places Malawi, Vietnam and India. An effort was also made in Cuba, but failed. The DHIS 2 project consists of four nodes or teams: Norway (usually referred to as 'Oslo'), Vietnam, India and Ethiopia. The HISP teams are separate legal nodes with their own employees and local issues.

The leadership group of HISP controls the overall project and oversees the movement of people between the nodes. It also works to get funding to the project from international organisations like the EU and WHO.

The core DHIS 1 development competence is located in Cape Town, South Africa, where the two core developers are situated. The South Africa team has extensive experience with health information systems and therefore has a highly influential role as advisor also in the DHIS 2 project. They have been collaborating with the coordinators and developers of DHIS 2 to define DHIS 2 requirements and plan the direction of the DHIS 2 development. India has also strongly influenced the direction of the software development. After the first official release of DHIS 2, they have become the main target area for piloting the system (Nordal, 2006). The feedback and requirements from India has affected much of the development focus.

An important focus for HISP is to develop local competence. During the DHIS 1 implementation process HISP put a lot of effort into establishing local capacity at the nodes in the network to be able to achieve sustainability and foster growth. The competence building efforts for DHIS 1 was primarily focused on use and software administration, but for DHIS 2, building local development teams has been an additional goal.

3.4 DHIS 2 development

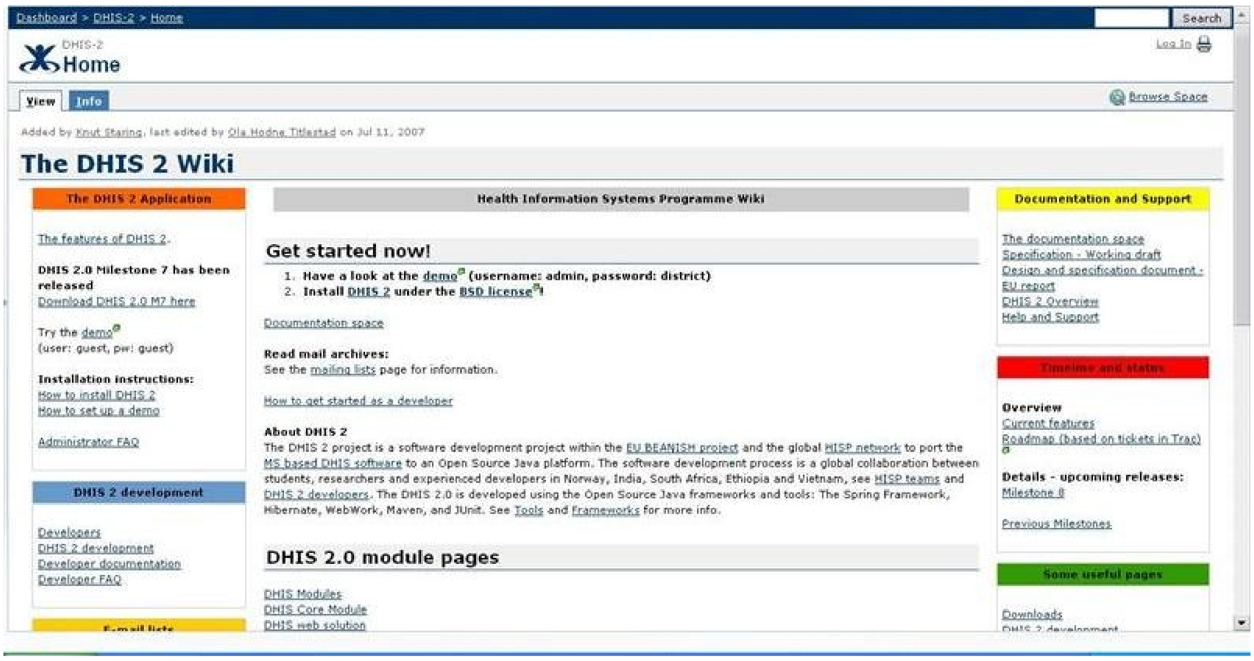
The technical and the social infrastructure in DHIS 2 will be presented in this chapter. The description provides essential background information for understanding the DHIS 2 case and helps to address the research objectives of this thesis.

3.4.1 Technical infrastructure in DHIS 2

The technical infrastructure of the HISP project consists of mailing lists, an online source code repository and several different types of web pages; a general wiki, country specific sites containing information regarding that particular country, a bug database and an online demo of the newest DHIS 2 release. The wiki pages covers both information about the HISP project in general and more specific information about the DHIS 2 software and the development process. The wiki also contains documentation which will be described in its own subsection.

3.4.1.1 Web pages

DHIS 2 has several different types of web pages, including wiki-pages, downloadable documentation, Javadoc and help functionality in the software.



Added by Knut Staring, last edited by Ola Hodne Titlestad on Jul 11, 2007

The DHIS 2 Wiki

The DHIS 2 Application

The features of DHIS 2.

DHIS 2.0 Milestone 7 has been released
Download DHIS 2.0 M7 here

Try the demo
(user: guest, pw: guest)

Installation instructions:
How to install DHIS 2
How to set up a demo

Administrator FAQ

DHIS 2 development

Developers
DHIS 2 development
Developer documentation
Developer FAQ

E-mail lists

Health Information Systems Programme Wiki

Get started now!

1. Have a look at the demo (username: admin, password: district)
2. Install DHIS 2 under the BSD license!

Documentation space

Read mail archives:
See the mailing lists page for information.

How to get started as a developer

About DHIS 2
The DHIS 2 project is a software development project within the EU BEANISH project and the global HISP network to port the MS based DHIS software to an Open Source Java platform. The software development process is a global collaboration between students, researchers and experienced developers in Norway, India, South Africa, Ethiopia and Vietnam, see HISP teams and DHIS 2 developers. The DHIS 2.0 is developed using the Open Source Java frameworks and tools: The Spring Framework, Hibernate, WebWork, Maven, and JUnit. See Tools and Frameworks for more info.

DHIS 2.0 module pages

DHIS Modules
DHIS Core Module
DHIS web solution

Documentation and Support

The documentation space
Specification - Working draft
Design and specification document - EU report
DHIS 2 Overview
Help and Support

Timeline and status

Overview
Current features
Roadmap (based on tickets in Trac)

Details - upcoming releases:
Milestone 8

Previous Milestones

Some useful pages

Downloads
DHIS 2 development

Figure 5: Screenshot of the DHIS 2 space on the wiki.

HISP uses Confluence from Atlassian as its wiki tool. Confluence is not open source, but Atlassian gives away free licenses to OSS projects that conform to a set of given requirements. Before deciding to use Confluence, it was discussed how appropriate it was to use such a "semi-proprietary" solution (Nordal, 2007). Confluence was chosen because of the quality of the product, the foothold they have in the OSS Java community, and because it is based on the same technologies being used in the development of DHIS 2 (ibid).

The Confluence wiki is divided into different sections called spaces, which are independently managed wikis, all part of the same site. The wiki for HISP is available at http://www.hisp.info, with a dedicated space for the DHIS 2 project at http://www.hisp.info/confluence/display/DHIS2. The wiki is the main tool for publishing information about DHIS 2 on the web. Anyone can sign up for an account and edit the pages. Local HISP web pages with more static information have been introduced in several countries, including the countries where DHIS 2 is being used, India (http://www.hispindia.org) and Vietnam (http://www.hispvietnam.info). Both these sites contain some information about the project, the Indian site written in English and the Vietnamese site in Vietnamese, but they are both unfinished sites under construction where not all information is presented.

Only registered users on the wiki are allowed to edit and create new pages, and only administrators are allowed to delete pages and comments.

3.4.1.2 Visual documentation

DHIS 2 had some diagrams in the source code earlier, but since it was agreed that diagrams do not belong in the source code, they were deleted. The old diagrams are still available on the wiki, but they are outdated and new ones should be drawn and made readily available on the wiki. Another form of visual documentation that is not presented on the DHIS 2 wiki is screenshots from the software. Fogel (2005, p. 27) argues that a single screenshot can be more convincing than descriptive text and output from mailing lists. Screenshots also show is functional and that the software actually works (ibid).

Visual documentation in the form of diagram, especially on the API, has been requested on the mailing list, but new ones have not been made.

3.4.1.3 Mailing lists

The mailing lists of DHIS 2 have been extensively used and serve as a common place for everyone involved to speak their mind and ask questions. They provide a great opportunity to conduct asynchronous communication between participants in different countries and time zones.

DHIS 2 originally started out with 4 mailing lists called dev, scm, jira and despots (Nordal, 2006). “dev”, dhis-dev@hisp.info, is a developers' mailing list and is the most important mailing list in the development of DHIS 2. The list brings developers from various nodes together to discuss everything regarding the development of DHIS 2; technical solutions, questions, support, coordination and so on. Developers and administrators with questions ask them on this list. In the rest of this thesis the developers' mailing list is referred to as the “dev-list” or just the “mailing list”.

“scm”, dhis-scm@hisp.info, is for mail automatically generated by Subversion when a participant commits an update to the central source code repository. Most, if not all developers who are subscribed to the dev-list are also receiving the commit mails.

“jira” (no longer in use) is for mail from JIRA, the former issue tracker, and “despots”, despots@hisp.info, is for administration purposes. After the first milestone release in January 2006 an additional mailing list named “user” was created to allow users of the software to give feedback and get help. When the infrastructure for mailing lists was established, a mailing list for the stable DHIS 1.4 was also created. Today, 4 mailing lists are commonly used and made reference to. The mailing list called JIRA has been removed and one new mailing list called “HISP”, hisp@hisp.info, has been established. The HISP list covers general topics regarding the HISP project. Additionally, there are a couple of more mailing lists for people interested in specific topics or located in a specific country. DHIS 2 developers are required to subscribe to at least the dev-list and the scm-list.

The mailing lists are not only for interaction between participants, but also serves as a complement to the documentation and knowledge transfer process (Nordal, 2006). This is especially important in hectic periods when things change overnight and keeping static documentation up to date would create too much work. To enhance the usefulness of the mailing lists and make it easier to refer to previous discussions, the mailing lists are stored in web archives.

In the beginning of 2007 it was decided to change to a new mailing list system which enables better archives and the possibility to search the archive. The system is also easier to administrate. The old archives still exist to prevent data from being lost, but they are no longer being updated since the mailing lists are closed.

3.4.1.4 Public code archive

Fogel (2005) state that everyone will expect an open source project to be using version control tools, and it was decided early in the development process that DHIS 2 needed a public code archive where all participants could contribute (Nordal, 2006). Both CVS and SVN were considered, but since CVS was the most widely used version control tool in open source projects at that time and was considered more mature then SVN, CVS was chosen for the project (ibid). It later turned out to be problems with CVS, especially regarding permission handling, and it was decided to change to Subversion (ibid). SVN has proved to be very stable and has been successfully used in the development of DHIS 2 since the beginning of 2005.

As of May 2007 there have been more than 3300 commits (changes done by developers) to the source code repository, and SVN is used nearly on a daily basis (98 out of 120 days in the first 4 months in 2007 had a commit) and by all developers. There were 634 commits to the repository from January to April 2007 - more than 5 commits every day on an average basis. This makes SVN one of the most frequently used tools in the development of DHIS 2, and it is essential for the collaboration between developers.

It is possible to anonymously access the source code over http[7], but only read access is granted. To be able to contribute to the project and commit new code, the participant has to sign up for an account. Anyone interested can do so, and how to do it is explained on the wiki.

When participants commit their code to the repository they are asked to provide a log message explaining the changes they have done. This log message, combined with a log message showing the differences between the new and the old source code, is automatically sent to the scm mailing list. All developers are encouraged to read the log messages to know what is going on in the repository and what the other developers are doing. The mailing lists show that people are paying attention and read the log messages. If there is a commit without a log message, it will not take long before someone sees it and writes a mail complaining about the missing log message; "*Please write log messages!*" The same goes when participants commit files that should not be in the repository, e.g. binary files, or do something else that seems out of place.

Several open source projects do not give write access to the source code right away and/or have trusted project members that control every commit. All the projects at Apache operates under a meritocracy where those participating to a high extent are invited to the project as a committer.[8] This is not the case with DHIS 2, where everyone involved in the project can commit to the central repository. The fact that everyone has write access to the source code could potentially lead to abuse of the repository, but has never been a problem in the development of DHIS 2. Another more important aspect, that could be a problem, is that no one is in charge of controlling the commits. This could lead to poor code or defect functionality, but by having the SVN server send log messages to a mailing lists with reports on all activities and encouraging people to read the mails, the project gains a measure of commit-control. Since DHIS 2 is a small project, whit less than 10 active developers, not having approval of the commits have not been a problem and most of the commits are without problems. Whether or not this

7 Http-access address: http://www.hisp.info/svn/scm

8 See the Apache site for more information: http://www.apache.org/

is sufficient commit-control has never, to my knowledge, been discussed among the participants.

3.4.1.5 Issue tracker

DHIS 2 has a bug and issue tracking software. DHIS 2 used for a long period of time JIRA from Atlassian since it's integration with the Confluence wiki provided benefits for the project. However, this integration possibility has not been used, apart from a common user database between the two tools. The core developers did actively use JIRA in the beginning, but problems with mail configuration in JIRA resulted in no mails being sent to the dev-list when changes were made (Nordal, 2006). This made it hard to follow what was going on, and as one of the project members wrote to the dev-list: "[...] *JIRA should send e-mails when changes occur. It is basically useless as anything more than a task list for individual developers in the state it's in now*".[9] Less mail activity leads to less use of JIRA, and new developers never got to see how JIRA might work and the usefulness it, or other bug and issue tracking software, can provide.

There are several reasons why the issue tracker was not used as much as it should be in the development of DHIS 2, but the most likely reason was the lack of e-mail support. Fogel (2005) mentions that it is important to have the tracker connected to a mailing list. It is hard to keep up to date and follow the development when no mails are sent from the tracker. The DHIS 2 developers could log into JIRA every now and then to see what is going on, but it is not a perfect solution nor sustainable in the long run. The lack of mail notification was also the root to the problem of getting new developers to use it (Nordal, 2006).

Other reasons why JIRA was not used include that the software is too complex for regular users and therefore only usable for developers. As mentioned earlier, Goldman and Gabriel (2005) stress the fact that the process of reporting bugs should be easy and not painful. The developers and coordinators involved in the DHIS 2 development has suggested several possible ways of how to get the users bugs and request into JIRA.

It was decided from the start that the issue tracker required user accounts to see and file issues (Nordal, 2006). The registration is open to anyone, but it requires that extra step of registering an account before it can be used. The reason for requiring an user account is to minimize the risk of abuse and to be able to contact the person who files an issue (ibid). Since the process of reporting bugs should be easy, requiring the users to create an account before reporting bugs could scare them away. DHIS 2 see little (or none) issues being filed from the end users, but as few of the end users has internet access or write English, the project is not likely to get issues directly from them either.

Goldman and Gabriel (2005) argue that issue trackers should not necessarily require that users register an account, and the observations from the DHIS 2 project suggest that this view could be reasonable. Findings from Nordal's thesis also observe how the value of the tools, and especially the issue tracker, diminishes when fewer participants use them. Nordal further suggests that when a project grow to a size where it is difficult to keep track of what everyone is doing, participants should be encourage or even in some situations demanded to use them.

9 From the mailing list archive: http://www.hisp.info/archives/dev/msg02388.html

3.4.1.6 Releases

Fogel (2005) mentions two popular systems to stabilize a release; a dictatorship model with a release owner and a more democratic vote system. DHIS 2 uses neither of these, but a democratic system where developers, coordinators and facilitates are given the right to tell their point of view. The release process in DHIS 2 usually starts with a discussion about what goes into the release, who is doing it and when it is to be released. Then the developers do their tasks and when the release date approaches one out of two things happens; either someone ask for more time to finish his or her task, and are granted that extra time, or the release date passes by without a release. It has been discussed several ways of doing the release process on the mailing list. Most people (in fact everyone who answered to one of the threads about releases on the mailing list) agreed that the best approach is to set a feature-freeze date and make a release branch. An excerpt from the discussion on the mailing list shows what the developers think:

> *XX days before the release we make a new branch/tag without changing the version number (still -SNAPSHOT). Bug-fixes go into this new branch and gets merged back to trunk. On the release date this branch is made ready for release like we normally do it (change versions).*[10]

This fits well with the statement of Fogel (2005) and Goldman and Gabriel (2005) that code freezing is not a good idea, but that using a release branch is. By having a release branch developers can continue on the features they either did not finish in time or that are scheduled for a later release, while bug fixing is happening in the branch. This sounds reasonable, but the problem is that it never has happened in the developing of DHIS 2. The new features in DHIS 2 are tested before a milestone release by voluntaries, but there are no formalize testing routines. The last milestone release (as of July 2007), milestone 7, was not tested probably before the release, and when the users downloaded it, they found several critical bugs that had to be fixed before they could deploy it in the field. This lead to a maintenance release a couple of weeks later. By either making a release branch a week or so before the release, or release a beta version, this kind of problems might not occur that often. The developers are aware of the problem with lack of testing and it is often raised when the next milestone is discussed, but so fare no one has step forward and actually changed the release process.

Another problem DHIS 2 faces is to agree on a release date and stick to this date. "*I don't think enough things have been committed to warrant a release right now, so suggest we put it off a bit.*"[11] Statement like this from a coordinator on the mailing list shows that no one is working towards a agreed release date or are sticking to the predefined date. One of the developers wrote to the mailing list argue that "*As a dev[eloper], I like writing code, I like writing new stuff, and I'd like to make everything perfect and polished before releasing it*".[12] This goes for most of the developers and can lead to delay in the release process when there are no release manager or owner, and no prioritized list of what goes in the release. A roadmap with a feature list (although, not a prioritized list) has been made and updated since this discussing.

10 From the mailing list archive: http://www.hisp.info/archives/dev/msg01707.html

11 From the mailing list archive: http://www.hisp.info/pipermail/dhis-dev/2007-May/000796.html

12 From the mailing list archive: http://www.hisp.info/archives/dev/msg03187.html

Erenkrantz and Taylor (2003) argue that if a project does not have a coherent release process, it may have problems attracting users or achieving a reputation for stability.

3.4.1.7 Other tools used in DHIS 2

A number of other tools are also being used in the DHIS 2 project, including an IRC channel, a blog and instant messaging clients.

An IRC channel is mainly designed for many-to-many communication in discussion forums and is a place where users and developers can ask each other questions and get instant responses (Fogel, 2005).

A blog (or weblog) is an online journal with entries, often known as posts, on whatever topic or topics interest the author (Goldman and Gabriel, 2005).

Instant messaging (IM) is a form of real-time communication between two or more people based on typed text (Wikipedia, 2007b). It is widely used between project members of DHIS 2, and all the developers has one or several accounts. Different IM clients are popular in different countries, and both GTalk, MSN, Skype and Yahoo! Messenger are being used.

3.4.2 Social infrastructure in DHIS 2

3.4.2.1 Leadership

DHIS 2 has a loosely organized project structure, where everyone involved, as in most open source project, is participating on a public mailing list and uttering their thoughts and point of view. The leadership or maintainer of DHIS 2 has mainly consisted of 2 coordinators and a core group. The core group is a subset of the participating developers and consist of a 3-5 developers who contribute to the project frequently and substantially. The core developer group is given decision-making authority in the development project. Who the core developers are, is not formalized or written down anywhere.

This sort of developer organization are usually reflected in the code repository where everyone has read access, but only core developers are allowed to directly modify the source tree (Cubranic and Booth, 1999). With DHIS 2 you need an account to get access to the repository, but there is not made any differences between the core group and the other developers when it comes to directly commits. This means no one is given the task of watching the other's commits and approve them, but the developers still pay attention to what goes into the repository through the commit mails.

3.4.2.2 Decision making

> *If people feel that they are involved in the decision-making process and that their viewpoints are heard and respected, then the community will generally accept whatever decision is made.*(Goldman and Gabriel, 2005).

The development of DHIS distinguish it self from most other OSS developing process by not being developed by the user of the software, but mainly by students who wants to participate in an open source project or other participants engaged to develop a system for someone else. Since DHIS 2 is a system for gathering health information, some knowledge about the health sector is needed to make the right decisions in some cases. Most of the decisions are open for discussion on the mailing lists, but even if consensus is reached among the people participating in the discussions, there are often uncertainties about the legitimacy of the decision (Nordal, 2006). The average developer lack a good overview of health information domain, and can therefor not always make informed decisions when if comes to health related topics, but needs input from the coordinators or the health personal in a given country.

When developers can not make all the decisions on their own, it can lead to delay in the development process. The developers may want to contribute and write code, but before they can go any further with the implementation or a problem, they might have to wait for input from participants who know the health sector and can tell them how a particular thing should be done or what the normal procedures in a given setting is. The coordinators of DHIS 2 have a deeper understanding of the health sector and pay attention to the dev-list to answer question which the developers can not know. One example from the dev-list where a developer ask a question about something he is working on, but do not know the answer to: "*...the Indicators don't have short names. Should they?*".[13] The developer has the technical competence to implement the request, but lacks the knowledge about the general demands of the use of DHIS 2, and whether this is something a health information system needs. One of the coordinators answered this question, and the short names where then implemented. My experience with DHIS 2 is that this is seldom a huge problem, but can causes some delay if the developers have to wait on someone with knowledge to answer a question.

To avoid more delay then necessary, it has been suggested to have the discussions tied to the health domain on a more general basis without technical terms. The reason is to get more people from the HISP network involved, and hopefully with more people participating, and especially people who knows the health domain, the faster and better the discussions and answers will be.

Decision making about technical topics are usually left to the developers to figure out, but everyone is more than welcome to participate in all the discussions. The developers have a good overview of the technology used in the system, and are given full decision making authority.

Some discussion on the mailing lists are quickly decided by the coordinators and are not left to further discussion. The discussion about how to write the name of the software is one example of a discussion that was decided quickly:

> *To avoid a new long not-very-important discussion that takes away important development time, [coordinator 1] and [coordinator 2] decide that the name is DHIS 2 for general use, and that each version has the name DHIS 2.0, DHIS 2.0.15, DHIS 2.1 etc.*[14]

What seems to be the biggest problem when it comes to decision making in the development of DHIS 2, is from my experience, to know when a decision or conclusion is reached. Some discussions on the mailing list simply stop at some point without a conclusion, other discussions have no, or few answers, and sometimes two participants

---

13 From the mailing list archive: http://www.hisp.info/archives/dev/msg01984.html

14 From the mailing list archive: http://www.hisp.info/archives/dev/msg02616.html

can not agree upon a given topic. It has been said that *“Sometimes it seems we're running some kind of strange and ineffective democracy via the mailing list.”*[15] DHIS 2 lacks guidelines telling when the developers have consensus to deal with an issue or when a decision is reached. It has been suggested on the mailing lists that when a feature request or issue is posted to the mailing list, the discussion should conclude with someone saying either “yes, we will do this” or “no, we are not going to do this”. But as with several other discussion, this one died out and no one followed up what was discussed. A related topic occurs when a conclusion is reached, but not registered in the issue tracker.

Nordal (2006) point out in his thesis a difference between the decision making process in DHIS 2 and that frequently found in other projects where a formal voting mechanism is used to fall back on when consensus cannot be reached. If this is the case, one with decision making authority, often the coordinators or leaders, must act as a benevolent dictator and make the final decision (ibid).

3.4.2.3 Task assignment

Several different groups of participants are working on the DHIS 2 project which has lead to various possibilities for task assignment. Students taking the HISP course are assigned task as part of their group work.

Task assignment to new developers in the development of DHIS 2 is usually done one out of two ways; If the new participant has taken the HISP course at UiO they are often put to continue the work they did during the course. If the developers either have not taken the course or want something else to work on, they are usually appointed a task from one of the coordinators. The task the established developers take on are usually determined by a negotiation process where the coordinators ask the developer to do a task, usually with a positive response, or the developer comes forward and appoint themselves to do the work. Some task are easily agreed upon, like this task assignment on an instant messaging conversation where one coordinator is trying to assign a developer to help another developer solving a task[16]:

Coordinator 1 says:
  Developer 1, can you help Developer 2 to integrate CDE into the data entry module?
Developer 1 says:
  sure
Coordinator 1 says:
  great

This kind of task assignment has also resulted in a sort of responsibility for modules or module owners. One module in the software is usually written by one, or occasionally by two developers. This developer then have a good overview of the module and the source code, and can answer question related to the module and the use of it. None of the task assignment process or the module responsibility are formalised in any document.

15 From the mailing list archive: http://www.hisp.info/archives/dev/msg02199.html

16 From a conversation available on the wiki: http://www.hisp.info/confluence/pages/viewpage.action?pageId=11790

The hired teams in Vietnam and India are HISP employees dedicated to the development and implementation of DHIS 2. Task assignment to these groups have primarily been target at local needs to where they are situated (Nordal 2006). The teams have been given specific modules or task to develop, with a given requirement specification worked out in collaboration effort between the Norwegian node and the local coordinators. The employed development teams are not that different from other developers, and they have been encouraged to take part in the discussion on the mailing lists and be a part of the open source community at the same level as any other participants.

3.4.2.4 Developer guidelines

The developer guidelines of DHIS 2 consist of a wiki section called “Development standards and conventions”.[17] These wiki pages contain a description of which standards, principles and guidelines to stick to when developing, including code conventions to improve maintainability and readability of the code. The page also contain downloadable formatting templates for integrated development environments (IDEs) to make it easier to stick to these code conventions. The developers have gotten used to these guidelines, and if someone commits code to the repository that do not follow the guidelines, they are likely to receive an e-mail on the dev-list asking them to correct their commit.

DHIS 2 also have guidelines for the mailing lists which tells how to write and not write e-mails (e.g. write in English, use plain text, do not include large attachments, answer below the question and so on).[18] Some of these rules are common sense, like write in English, other are based on traditions from newsgroup and mailing lists, like respond below the text you are replying to, use plain text and avoid acronyms like “plz” and “thx”. Not everyone is aware of these rules, especially non-technical participants, resulting in some comments on the mailing list to follow the guidelines.

> *Just one thing: Please don't answer on top (like I am doing now), but rather below the text you are replying to (while cutting away unnecessary parts of the conversation).*[19]

Nordal (2006) conclude in his thesis that formal guidelines for how the participants should interact with the information management infrastructure should be in place to avoid that unstructured information generates an unnecessary and frustrating burden for the participants.

3.4.2.5 Milestone releases

Before making a release, it must be decided what goes into the release and what is not yet ready. DHIS 2 has a roadmap on the wiki[20] telling when the milestones are to be released and what it should contain, but there are no guidelines to how the release process is actually done.

17 Available at http://www.hisp.info/confluence/display/DOC/Development+standards+and+conventions

18 The guidelines from the mailing lists page http://www.hisp.info/confluence/display/DOC/Mailing+lists

19 From the mailing list archive: http://www.hisp.info/archives/dev/msg03847.html

20 The roadmap is available on the wiki: http://www.hisp.info/confluence/display/DHIS2/Roadmap

DHIS 2 is released in milestone releases, with M1, M2... and M7.1 meaning milestone 1, milestone 2 and milestone 7 maintenance release 1. DHIS 2 started to use 1.0.0-SNAPSHOT versions because they were working toward the 1.0.0 release, which was originally planned to be released at the end of 2006. This did not happened, and the new release date has been purposed to December 2007. There are no formal description of what is needed to make it to a 1.0.0 release, but one rule of tomb has been to provide the same functionality as in DHIS 1.4.

In the beginning it was decided to have a monthly release the 15th every month. What was going into each release was decided among developers and coordinators up front. This was successful for the first 4-5 milestone releases, but after the fifth release, nothing happened for a very long time. The developers were still developing and the software was deployed several places in Vietnam and India, but there was no new milestone release for months. Following the roadmap, milestone 6 should have been released July 15, 2006, but was not released until December 13 (see Table 2: Release dates and scheduled release dates for the milestone releases).

Milestone	**Scheduled release date**	**Actual release date**
M1	2006–02-15	2006–02-15
M2	2006–03-15	2006–03-17
M3	2006–04-15	2006–04-17
M4	2006–05-15	2006–05-15
M5	2006–06-15	2006–06-15
M6	2006–07-15	2006–12-13
M7	2007–03-15	2007–04-02
M8	May 2007	Not yet released

Table 2: Release dates and scheduled release dates for the milestone releases

After the release of milestone 6 it was decided to not have monthly releases, but to give the developers more time to make a bug free release with more functionality. The first release in 2007, milestone 7, was scheduled to March 15, but did not happened until April 2. The process of updating the roadmap and distribute tasks were done among developers and coordinators from Norway, Vietnam and India, using instant messenger clients. This was done for each of the milestones, before the actually developing started.

The releases are available as both user-friendly and developer-friendly release formats and can be downloaded from the wiki.[21] The download page contains a WAR file to be used by administrators who want to install and run the software, a demo WAR file containing data to be used as a demo, and the source code in different file format. The Vietnamese team has used Installer2GO, a tool to make installation packages for Windows, when they have deployed DHIS 2 in their districts, while India has used a similar tool, InstallShield.

21 The DHIS 2 download page: http://www.hisp.info/confluence/display/DHIS2/Downloads

3.5 Is DHIS 2 an open source project?

Although coordinators and other participants often make references to DHIS 2 as an open source project, it is not always clear that DHIS 2 in fact is an open source project. Nordal (2006) argues in his thesis that DHIS 2 is not a pure OSS project, but share several similarities to hybrid projects like the Mozilla web browser. Although there are some similarities, he also describes some differences. DHIS 2 does not have the strong financial backing which is found in hybrid project and no guarantees for the amount of time and effort the developers will spend on the project.

A common denominator in OSS projects is that the developers also are the user of the software (Gacek and Arief, 2004). The developers are then intimately familiar with the features and know what the correct and desirable behavior of the software is (Mockus 2000). The developers of DHIS 2 are not the user of the system, and according to Mockus (2000), the developers are unlikely to have the necessary level of domain expertise when they are not also experienced user of the software, and will unlikely have the necessary motivation to succeed as an OSS project. Eric Raymond also states in his well know paper, “The Cathedral and the Bazaar”, that “*Every good work of software starts by scratching a developer's personal itch.*” In the developing of DHIS 2 it is not the developers personal itch that get scratched, at least not in the way that they develop a software they use themselves. The developers of DHIS 2 develop a software that is used in the health sector in developing countries, and the developers have no use for it's functionality themselves. As seen earlier the motivation for participate in OSS project are numerous and complex, and the personal itch the developers of DHIS 2 scratch can be to gain knowledge, help out in developing countries or other issues at a personal level

The way a project gain new participants are usually influenced by the same aspect, but not in the DHIS 2 project. Most developers from Norway gets introduced to the software and the project through the course at the University of Oslo, and are later becoming a part of the development team. In India most developers are hired through the HISP India project. DHIS 2 wants to get participants from outside the HISP network, but Nordal (2006) suggest that HISP is not doing enough to attract these outside participants. When the DHIS 2 project first started it was put a lot of effort into having flexibility in the core of the software to enable this part of the software to be used outside the health sector, and to appeal to a larger group of participators (ibid). Experience from the DHIS 2 project suggest that it takes a lot more than publishing source code on the Internet and having an open development process to get new participants (ibid), and as of July 2007, DHIS 2 has not seen any outside participants yet.

The DHIS 2 project use a lot of the tools, functionality and infrastructure usually found in OSS project like a freely available software, mailing lists, wikis, public code archive, version control system, geographically distributed developers etcetera. This does not automatically make DHIS 2 an open source project, but the fact that it is released under an OSS license, makes it OSS in a legal aspect. The single most important requirement of an OSS is that it's source code is freely available to anyone who wishes to examine or change it (Godfrey and Tu, 2000), which DHIS 2 is. Combined with the fact that the project managers and other stakeholders define DHIS 2 as an open source project, makes it obvious to me to use literature from the open source world in this thesis.

4 Methods

This chapter presents the methods I have used during my research along with a description of my research approach.

4.1 Action research

The research approach used in this thesis falls within the framework of action research (AR). Put simply, action research is “learning by doing”. A more complete definition is given by Gilmore, Krantz and Ramirez (cited in O'Brien, 2001, What is Action Research?):

> *Action research...aims to contribute both to the practical concerns of people in an immediate problematic situation and to further the goals of social science simultaneously. Thus, there is a dual commitment in action research to study a system and concurrently to collaborate with members of the system in changing it in what is together regarded as a desirable direction. Accomplishing this twin goal requires the active collaboration of researcher and client, and thus it stresses the importance of co-learning as a primary aspect of the research process.*

AR is thus a process of social research where both outsiders and problem owners work together to solve problems and reach common goals. Action research refers not to a single, monolithic research method, but rather to a class of research approaches. Baskerville (1999) list four common, agreed characteristics of AR:

- An action and change orientation.
- A problem focus.
- An “organic” process involving systematic and sometimes iterative stages.
- Collaboration among participants.

AR is a collaborative research network where democracy has a central role. Each participant is a co-researcher and every participant's ideas are considered equally significant as potential resources. An AR researcher will work with the problem owner to gain as much knowledge on the problem domain as possible and insights that cannot be understood by studying it from a distance. This is in contrast to conventional social research, where the researchers try to be fully objective and thus are not interacting with what they are studying.

O'Brien (2001) points to several other attributes of AR that separates it from traditional interpretive research. Primarily, AR focuses on turning the people involved into researchers. People learn more, and use what they have learned more willingly, when they do it themselves. AR also has a secondary social dimension; the research takes place in real-world situations and aims to solve real problems. Finally, the researcher makes no attempt to remain objective.

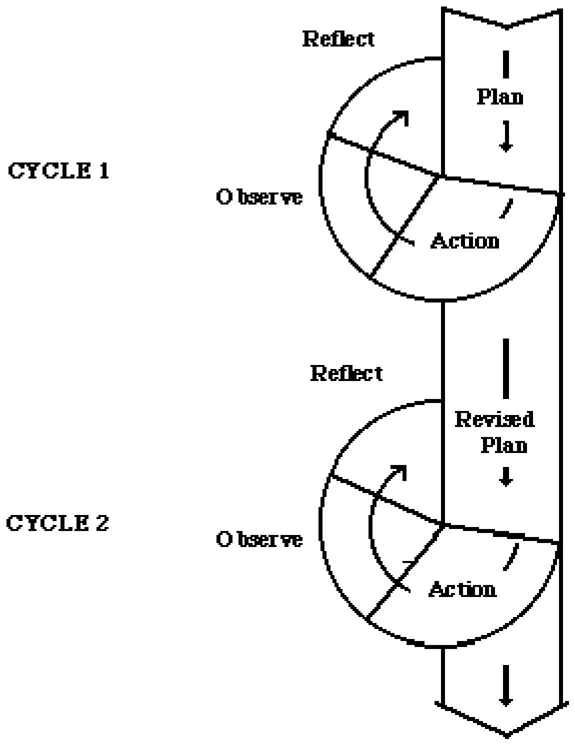


Figure 6: Simple Action Research Model (from MacIsaac, 1995)

Kurt Lewin is generally considered the 'father' of action research. He first coined the term 'action research' in his 1946 paper "Action Research and Minority Problems". He characterizes Action Research as "a comparative research on the conditions and effects of various forms of social action and research leading to social action" (ibid). AR is typically preformed as an iterative process where the researcher refine the interventions (actions) on a cyclical basis. Each of these cycles have four steps; plan, act, observe and reflect (see Figure 6: Simple Action Research Model (from MacIsaac, 1995).

Initially, the problem is identified and one or more possible solutions to the problem leads to an action plan which is then implemented. During the implementation, data on the results is collected, analysed and observed. These findings are studied and interpreted with regards to how successful the intervention was. Based on the reflection, a plan for action is re-assessed and refined, and a new iteration can begin. This cycle continues until the problem is solved.

4.2 Action research in the field of IS

Action research has been an established research method in the social and medical sciences since the mid-twentieth century, but it was not until toward the end of the 1990's that AR began to grow in popularity in IS research fields (Baskerville, 1999). Following The International Federation for Information Processing conference in 1998, Avison et al. (1999) reported that AR has now gained acceptance at the same level as quantitative studies in the field of IS. They pointed to five main contributions of AR in development of information systems;

- The Multiview contingent systems development framework
- The soft systems methodology
- The Tavistock School's socio-technical design
- Scandinavian research efforts intended to empower trade unions
- The Effective Technical and Human Implementation of Computer-based Systems (ETHICS) participative and ethical approach to information systems development

The shift to qualitative methods by the mainstream of researchers was manifested in 2003 by a special issue of the prestigious paper MIS Quarterly named "Action Research in Information Systems".

4.3 My research approach

In this section I will outline my choice of research approach. Following the tradition of HISP, action research has been used to conduct the research work both in Vietnam and in Norway. I will also present the HISP team I have worked with in the project and the methods used to obtain the data.

4.3.1 The HISP team

The HISP team supervising the development of DHIS 2 consists of two coordinators at the University of Oslo in Norway, who are PhD students at the Department of Informatics. One of the coordinators has, since the fall of 2006, been living in Zanzibar to work with the local HISP team there, but still functions as a coordinator for the overall HISP project.

The leadership and coordination efforts are mainly centralized at UiO, but independent development teams have been established in the countries where HISP is piloting DHIS 2, like HISP India and HISP Vietnam. These teams have their own coordinators who are working in close collaboration with and being supervised by the coordinators in Norway. Norway does not have its own

The two developers of DHIS 1 are both hired by HISP South Africa, while the developers of DHIS 2 mainly consist of master students from Norway. In addition, there are hired employees in both Vietnam and India.

The DHIS 2 project is a part of the larger HISP network, and the development effort has benefited from the large number of professionals involved in HISP. These range from medical doctors and health workers to software developers involved in the global HISP network.

I was introduced to HISP and DHIS 2 in a university course in the autumn of 2005, and took slowly more and more part in the project after that. It was not until I arrived in Vietnam in the summer of 2006 that I begun taking part in DHIS 2 development in earnest. Since then I have been a part of the development team.

4.3.2 Interviews

DHIS 2 is a global project with people situated in different locations around the world, and the participants I have collaborated with have often been working from other locations than myself. As a result, combined with the tight schedule for most people involved, few formal and planned interviews have been conducted. Instead, there has been informal conversations and constructive discussions with participants at all levels in the project.

These conversations have either been done face-to-face, privately on instant messaging clients or e-mail, or on the open mailing lists. The mailing lists have functioned as the project's main communications channel, and is a place where everyone involved can speak their mind, making it a place to get different points of view on any given topic.

The face to face conversations have been mainly day to day communication with people I worked with, but also more structured discussions where we have sat down to discuss particular subjects. The strong focus on participation in action research has also made the informal approach a natural way for me to gather data.

4.3.3 Development and participation

I have participated in the development of DHIS 2 by developing a module for making validation rules. Although the actual work of writing the code was done alone, the requirements and underlying thinking and decision making is based on cooperation and discussion with the other participants.

In Vietnam, everything regarding DHIS 2, from implementing, testing and installing the system to future planning, was conducted in cooperation between the Norwegian project members in Vietnam and the developers and health service workers in Vietnam. In action research, the researcher works with the problem owner to gain knowledge of and insight into the problem domain that cannot be understood by studying it from a distance. I learned a lot about every aspect of the DHIS 2 development by being in Vietnam and working with the other participants.

4.3.4 Meetings

Meeting takes place at all levels in the HISP project; there are coordination and organisation meetings both locally and globally between participants, and formal meetings between HISP and the health ministry in given countries.

When I first arrived in Vietnam we had a meeting between the participants situated in Vietnam and one of the coordinators to discuss the situation and the further developing of DHIS 2 in Vietnam. Later, when we arrived in Hue, we attended a meeting between the health ministry in Vietnam and the HISP project where the future direction of the DHIS 2 implementations process in Hue was discussed. This meeting and its outcome determined most of our work and process while in Vietnam. Back in Norway I have attended less formal meetings where the future development plan or particular modules in the system have been discussed. Each of these meetings have given me a deeper understanding of and insight into how the DHIS 2 development and its decision processes function.

5 Empirical study

In this section I will give a topical overview of the empirical work I have done and taken part in during my work with the DHIS 2 project.

5.1 Introduction to the HISP project

I was introduced to the software and the development team through the HISP course at the University of Oslo in the fall of 2005. The course was held by one of the coordinators, and with three of the most active DHIS 2 developers as teaching assistants, I got a chance to know some of the participants face-to-face. I gradually took more and more part in the development community and the development process throughout the spring of 2006. By reading the mailing lists and talking face-to-face with the project members, I gradually learnt who the participants were and which module they were working on. Based on my impression of who the most active developers were, and who others referred to as core developers, I also learned who the core developers was.

The local HISP team[22] and the DHIS 2 development team[23] are presented on the wiki, but what the developers are currently working on, who the core developers are, and what it takes to become a core developer is not formalized anywhere.

5.2 My experiences from Vietnam

I spent 4 months in Vietnam, from July to November 2006 being a part of the Hue development and implementation team. This chapter describes my time in Vietnam and the work I did while situated there.

5.2.1 Arriving in Vietnam

I arrived in Ho Chi Minh City, the capital of Vietnam together with two other Norwegian participants in the summer of 2006. We knew little up front about what we were going to do while situated in Vietnam, but we knew that the majority of Norwegian HISP participants who had been in Vietnam before had worked with the four employees in Ho Chi Minh City (HCMC).

We were welcomed by the four developers at the airport when we arrived, and after being introduced to them, the city and the office where they worked, we all attended a meeting a couple of days later with one of the Norwegian coordinators visiting Vietnam. The coordinator talked about the future plan for implementation and development in the two HISP nodes in Vietnam, HCMC and Hue. It was decided that all three of the Norwegian participants and one of the Vietnamese developers should go to Hue to strengthen this node, which only consisted of one Vietnamese employee. One of the Norwegian and the employee from HCMC would return to HCMC after some time to continue the work from there. DHIS 1.4 had been installed and to some extent used in 5 districts in and around Hue, and one of our task was to upgrade to DHIS 2 in these districts.

22 HISP teams on the wiki: http://www.hisp.info/confluence/display/HISP/HISP+Teams

23 DHIS 2 developers: http://www.hisp.info/confluence/display/DHIS2/Contact+info

The first couple of days in Hue consisted mainly of meetings and getting to know the city, the people and the current situation. Both the coordinator of HISP Vietnam and one representative from the Ministry of Health were visiting Hue at the same time, and we attended several meetings and social events together. In one meeting, consisting of the coordinator of HISP Vietnam, the coordinator from Norway, the representative from Ministry of Health, the Norwegian HISP participants, the Vietnamese HISP participants and three others from the local health service, our tasks and the responsibilities of both parties were discussed. The meeting was held in Vietnamese, making it difficult to follow, but the outcome was a signed contract between HISP and the Health Service of Thua Thien-Hue province (see appendix A). It was also decided that we should be given a place to work at the health office in Hue, and that we were only going to install DHIS 2 in four out of five districts, since the last district is situated too far from the city.

The Norwegians and the developer from HCMC were all situated at the same hotel, which also served as our office for a period of time. We bought a wireless router, and had some tables and chairs brought to one of the room, making it an nice place to work.

When I first started my action research, the plan was to work on a module called Customized Data Entry. I had been working on a similar module, Data Entry, as part of my assignment in the HISP course the previous fall, and as a typical task assignment process in DHIS 2, I was asked to continue this work.

A plan for our work while situated in Hue was made at one meeting shortly after we arrived to Hue (see Appendix B). This plan also included some future tasks to be carried out for the Vietnamese participants after we had left. I was made responsible for the Customized Data Entry module with help from one developer in Norway, and for fixing issues and bugs found in the field. The following is an excerpt from the plan, showing my responsibilities:

6. Customized data entry module

Deadlines:

First sample report (B10)	September 1
Prototype of a generic tool	September 29
First release	November 30
Developer:	[the author]
Support:	[Developer in Norway]

4. Bug fixing and software improvements

- Based on user feedback (be active in retrieving feedback)

Responsible: [Norwegian participants in Vietnam], [the author]

5.2.2 The start up phase

I had a brief overview of the tools and technologies being used in the development from the HISP course, but the actual code had changed quite a bit since then, and there was a lot to learn, understand and get an overview of. The wiki pages of DHIS 2 contain little information about the tools being used in the development, besides links to additional information and tutorials. This is great when you want to learn about a tool or technology in isolation, but the main challenge for me was to learn how these tools and technologies interact with one another.

I found a tutorial on the wiki written by a student during one of the HISP courses called "Hello WebWork - Beginners tutorial from a beginner".[24] This tutorial explains how WebWork works with both Maven and Jetty, but unfortunately for me, the tutorial was outdated and the provided example did not work. I started looking at the given example and tried to make it work with the newest versions of the tools. After some struggle I got it running, and updated the text on the wiki page and provided a new working example. The task of updating the tutorial gave me a better understanding of how these selected tools and technologies interact, and I learned a lot from it.

Several tutorials or how-tos explaining how DHIS 2 works, have been written and made available on the wiki since then, including "How to build a period selector", "How to create an exporter" and "How to use the Organisation Unit Web Tree".

After a couple of weeks in Vietnam struggling with making a new web module for the Customized Data Entry, one of the coordinators in Norway asked me to implement functionality to generate minimum and maximum values based on previously entered values. I was told to look at the DHIS 1.4 version, where this functionality is implemented, and no further explanation was provided. I wrote an e-mail to the dev-list asking what the requirements specification for the min/max functionality should be, but no one answered. I then made up my own specification based on what was done in DHIS 1.4 and what I thought was the best and most logical way of solving the problem.

At one point I got stuck with a problem regarding the min/max functionality, and asked one of the core developers if he had any suggestion to how it could be solved. The core developer had solved a similar problem before, but could not recall how it should be done, and with no written documentation, it was a dead end. I eventually ended up doing the whole thing in a different way to bypass the problem.

Working on this small task made me more familiar with both the source code and the tools, and was a better way for me to learn, compared to creating a whole new module. With help from one of the other Norwegians in Vietnam, we solved the task together.

While working with the software, I discovered that it was possible to enter data element values below zero. I wrote an e-mail to the dev-list asking if this was correct, and was told that it was not. I was asked if I could work on validation instead of the customized data entry, since customized data entry was going to be developed by a student group. I had never heard about the validation module before, and I had spent a significant amount of time trying to understand the customized data entry and its requirements, making the sudden switch of plan a surprise to me. In any event, I said I could do it, and started concentrating on the validation module. My experience with the module and the development of it is described in section 5.3.

5.2.3 Development and installation

In one of the meetings we attended when we first arrived in Hue, we were promised an office to work from, but this was easier said than done. After a couple of weeks, the Vietnamese employee in Hue told us there was a problem of getting acceptance to work at the health centre. One of the Norwegians, later the same day, spoke with the coordinator in Norway in an instant messaging chat, and he was more up-to-date on the problem than we were. He even suggested that we perhaps should go back to Ho Chi

24 "Hello WebWork – Beginners tutorial from a beginner", available on the wiki: http://www.hisp.info/confluence/pages/viewpage.action?pageId=1532

Minh City since they apparently did not want foreigners working in their offices. Having the people back in Norway more up-to-date on the problem in Hue then we were sometimes made it hard to know what was actually going on.

In the first month in Vietnam I spent a lot of time working together and discussing problems with the other Norwegian developer in Vietnam. He went back to HCMC after a while to help out there and develop his own module, leaving me to work by myself. I still had the mailing lists and instant messaging chats with other developers to solve problems and discuss further development, but I enjoy and learn a lot by working closely with others.

The implementation process in the districts mainly followed the pre-made plan, but since the customized data entry module was not my responsibility any longer, I did not have any progress plan to follow. Making a new progress plan for the validation module was never discussed and it did not occur to me to ask for one either.

5.2.3.1 Installing DHIS 2

One of our goals while situated in Hue was to upgrade from DHIS 1.4 to DHIS 2 in four districts in and around Hue. I was not the one with main responsible for this task, but I participated by testing installation packages and helping out whenever necessary. I also came along on most of the trips to the hospitals to install, upgrade and teach the end users to use the software. Since this work is not within the scope of this thesis, it is not described any further here. Berg (2007) describes in his thesis some of the work and challenges around the implementation in Hue.

5.2.3.2 Collaboration in Hue

The collaboration with the Vietnamese employee in Hue was not that close. She came to the hotel and worked with us in the beginning, but she was mainly working on local reports which she could do by herself and without any help or input from us. She also had an office at the health centre, but since we were having problems getting permission to work there, we could not work together at the health centre. She was, after a period with little collaboration, told by one of the coordinators that she had to work at least three days a week together with the Norwegians. When we visited the hospitals in the districts, she was always with us, as she needed to be since none of the health workers speak any English. After a couple of months, we were finally given permission to work at the health centre, but the office had poor internet connection, no air condition and construction work was going on outside the window. We spent some time there, but the facilities were better at the hotel, making that our primary work place.

5.2.3.3 Interruption in development and installation

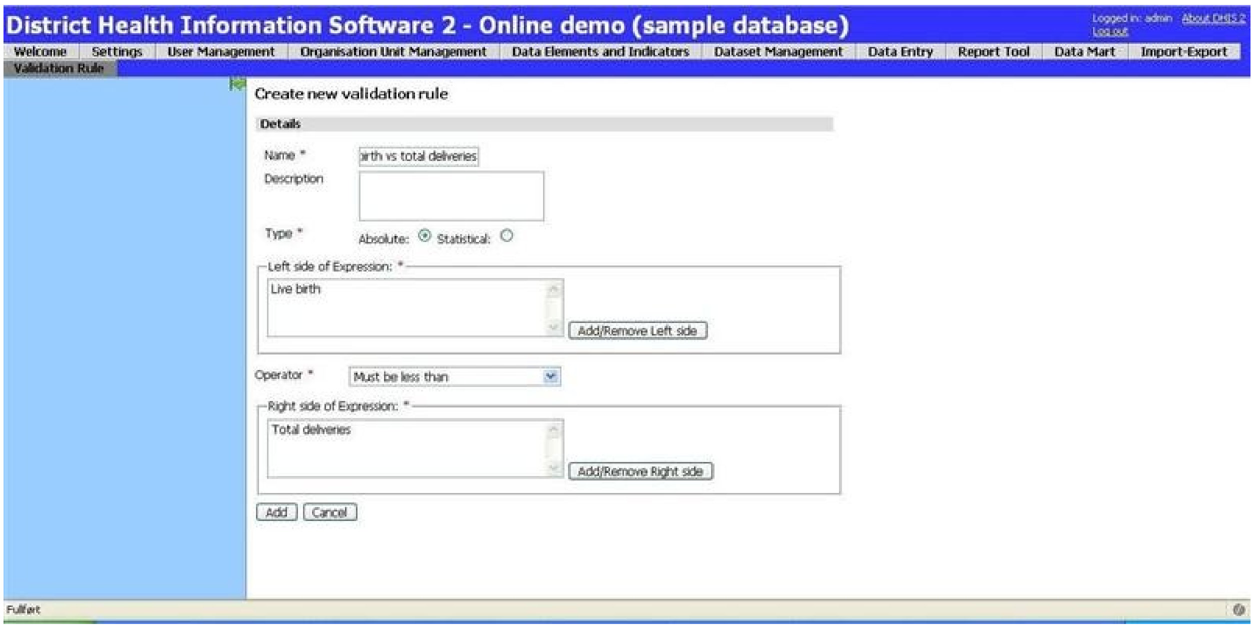
One annoying problem we encountered several times was malware; computer viruses and computer worms. We usually transferred data between our computers and the computers at the districts by USB flash drive. We did not think about the possibilities of malware in the beginning since it is seldom a problem in Norway, but after all the developer's machines, except the Mac, had been infected, we became more careful before moving files between computers, flash drives and external hard disks.

The viruses we obtained were not very harmful, but they were very annoying, making the computer reboot itself when writing particular words or phrases, hide folders, copying itself to all attached mediums etcetera. I spent one whole day removing the first virus from my machine.

After detecting viruses on the computers at all districts, we decided to install one of the best free anti-virus program we know about, AVG Anti-Virus. The computers at the districts are slow, making the virus scan take a long time, but we did manage to remove a lot viruses. Only one of the districts has internet access, making it hard to update the anti-virus program. We told them to be careful before copying files between computers, but they do not seem to know the danger of viruses and what they can do, even though we tried to explain it to them.

Another solution to avoid malware could be to install Linux, since most worms and viruses are written for the Windows operating system. For some unknown reason, we never really thought about changing the operation system at the districts. Although the health workers use some software that only runs on Windows machine, and installing and configuring Linux may be a bit harder than installing Windows, the possibility of running Linux should absolutely be considered in the future. Most Linux distributions are open source and free of charge, making them most suitable for developing countries.

5.3 The validation module



Create new validation rule

Details

Name * ırth vs total deliveries

Description

Type * Absolute: [✓] Statistical:

Left side of Expression: * Live birth

Operator * Must be less than

Right side of Expression: * Total deliveries

Figure 7: Screenshot from the Validation module in DHIS 2.

The validation module is a module to do validation on data elements in the system, where I have been responsible for the development. Data elements refers to objects which one can register data for in the software. For example, when counting the number of tuberculosis (TB) vaccinated patients in a district, "Number of TB vaccinated patients" would be a data element. The validation module contain predefined rules to ensure the quality of this data.

The first thing I did when I was suddenly asked to develop this module, was to write an e-mail asking what the requirements of the module was. I did not get any good answers besides “look at DHIS 1.4 and the documentation”. I was told that one developer had worked on validation before, and that there was some code for it in the repository. I found the code, but could not understand all of it, so I turned to the wiki to see what was written about the module. I found some documentation about validation, but it did not explain every aspect of it, and I did not understand why some things were done the way they were. I wrote an e-mail to the developer who had developed the validation part, but he could not recall what he had done and why he had done it the way it was. I was left with old code no one really knew what was doing or why it was done the way it was, supposed to develop a module I did not understand every aspect of.

I asked on of the coordinators if he could explain more about the concept of validation rule and how it was supposed to work, but was told he was not the right person to ask. He gave me e-mail addresses to the developer of DHIS 1.4, who answered my e-mail and explained some of the basic concepts about validation rule like when they are suppose to be used, what the difference between various types of validation rules are and the like.

I decided to start out with what I thought was the easiest part, the graphical user interface (GUI), before struggling more with the concept of validation rule. I used the 1.4 version as guideline and developed something similar, but adjusted to the version 2 context.

Finding it hard to figure out the best way to approach the module, I also spent much of my time writing and organising the documentation, which is another important task. This work is described in subsection 5.4.1. I also solved small issues I found in the software, helped out with the installation packages and the installation and fixed bugs based on feedback from the end users.

One of the first things I did when it came to the coding, was to delete parts of the old source code which neither I nor the other developers thought was necessary. I started originally out with using the rest of the old code, but after some trial and failure, I found it easier to rewrite everything from scratch. I discussed the structure of the validation module and its connection modules with one of the core developers, summarised in an e-mail to the dev-list.[25] I did not finish the module before leaving Vietnam, but continued the work back in Norway. In a meeting prior to the 7th milestone release, we discussed what parts of the functionality for the module should be finished for the milestone release and what could be postponed to later releases. At the time of writing, the module is still not completed, as work on this thesis has taken priority.

5.4 Technical infrastructure

As said earlier I have on several occasions found the documentation to be superficial, wrong, poor or missing. This section tells my experiences with and the evolution of the web pages and documentation from when I first joined the project and up to the summer of 2007. Topics relevant to documentation, like help functionality and mailing lists, are also covered.

25 From the mailing list archive: http://www.hisp.info/archives/dev/msg03198.html

5.4.1 Web pages and documentation

When I first joined the project, the wiki-pages suffered from a lack of structure and insufficient documentation. Prior to the first milestone release in the beginning of 2006, the developers worked very hard and documentation was given low priority due to high pressure from the Kerala state in India, to deliver the milestone on time .

I got involved in the project right after this milestone release and felt a lot of frustration when I did not find the information I needed or was looking for, because it was in a strange place or not present at all. Confluence has a search functionality which is an important tool for finding information on the wiki, but this functionality is not good enough. If you are in a space when searching, the default is to only search within that space. This may not be obvious to the users and can result in fewer or none search results because the information is located in another space. The shortage of the search functionality can be witnessed in the mailing lists, where some questions are related to topics that are already in the wiki.

As the first milestone was released, there was a discussion about documentation on the mailing lists. The discussion died out and nothing more happened. Several months later, the discussion was raised again, but with no change in the outcome from last time. An example of the frustration some new participants have felt when they enter the projects can be seen on the dev-list. This e-mail is from a new developer trying to understand the source code of DHIS 2:

> *I am trying to familiarise myself with the DHIS 2 source code. The developers documentation is outdated. It is not in sync with the current state of DHIS 2. The documentation confuses rater than explain, it is better with no documentation than wrong documentation.*[26]

Everyone I have interacted with agrees that documentation is important and should be updated and present, but no one actually did anything about it. As one of the coordinators wrote to the mailing list:

> *I think we all agree that documentation is very important. So what is needed is then a joint effort in providing this documentation. This is not always easy as we experience limited resources and a constant push from the users to release new functionality.*[27]

The developers and coordinators have a lot to do, and documentation is given a low priority. It was not until one of the developers in Norway in the autumn of 2006 founded “The DHIS Documentation project”, that things started happening. This project is discussed in the next subsection.

Maven site information was added to the repository in the beginning of 2007, and Maven generated websites were uploaded.[28] These generated pages contain general project information such as source repositories, dependencies and the Javadoc. The Maven generated websites has not been updated on a regular basis. According to the

26 From the mailing list archive: http://www.hisp.info/archives/dev/msg02774.html

27 From the mailing list archive: http://www.hisp.info/archives/dev/msg02408.html

28 The root project is available at http://www.hisp.info/site, and the web project can be found at http://www.hisp.info/site/web.

last published date on the sites, the web project was last updated on January 31, 2007, and the root project on February 3.

5.4.1.1 The DHIS documentation project

The DHIS documentation project was founded in the autumn of 2006 after an initiative from one developer in Norway. He started the work by making a separate space on the Confluence wiki called “DHIS Documentation”, and then announced the project on the mailing list.

A couple of people, including myself, indicated their interest in the project. I had been disappointed about the lack of documentation on several occasions, and was very pleased to see this suggestion be brought forward. I was a bit afraid it would turn out like many other suggestion where no one does anything and the discussion dies out. To prevent this, I decided to join in and do my part of the work.

The topics covered in the documentation space are listed in Appendix C. In addition to the ones listed in the appendix, pages with information about the documentation project and its participants have been created, but these have not been used yet. Other pages that are listed, but have not been filled with content as of this writing, include the user manual, the user FAQ, configuring DHIS 2, concepts and techniques, overview and connections of the modules and the system FAQ.

After the skeleton for the space was made, the actual writing started. I did a lot of the work moving pages back and forth, deleting outdated pages and started writing some of the new pages. After a page or section was written, an e-mail was sent to the mailing lists to get feedback from the other participants. One of the e-mail looked like this:

> *Please take a look at the documentation page for development environment and tools, http://www.hisp.info/confluence/display/DOC/Development+environment+and+tools, and let me know what you think. Is something missing? Or wrong?*
>
> *Feel free to make changes:)*[29]

Feedback from the other participants was then included and the documentation was improved until it reached a worthy level. I got positive feedback from some of the developers who liked the way the documentation was written and then presented on the mailing lists for feedback.

The pages in the documentation space are not regarded as finished after they are written and discussed on the mailing lists, as documentation is always a work in progress. However, with little time and few writers, the resources are used on getting as many pages as possible good enough, and then elaborate on the topics later on when the participants have time.

The DHIS Documentation project does not have any written rules or guidelines beside the given skeleton, and there are no formal members or leaders. There are a few people, including myself, who has contributed and written most of the information, but everyone can participate and write what they like.

29 From the mailing list archive: http://www.hisp.info/archives/dev/msg02987.html

5.4.1.2 The HISP wiki

Most of the work I did on the wiki was related to the DHIS 2 space and the new Documentation space. Some cleanup was also made to the HISP space and the Research and Development space, especially moving pages around in a better hierarchy and removing old, outdated material. Since I was not familiar with these spaces, I was afraid I might be messing up someone else's work if I did too many changes. One of the positive things about wiki pages is that it is easy to roll back to a previous version if needed, but to be absolutely sure I did not delete anything relevant; I made a space called “Deprecated” where pages I was not sure about were moved. An e-mail was sent to the mailing list telling the other participants about the space, and that if they found a page worth keeping there, they should move it back to where it belonged. As of July 2007 no pages has been moved out from the deprecated space.

Few participants contribute beside minor changes to the HISP wiki, and as one coordinator puts it on the mailing list:

> *A wiki is supposed to be a living "document" where everyone chips in to keep it updated (see wikipedia). So please let us all join in. For those who need some help to get going, pls don't hesitate to ask on the list - most HISP people have yet to contribute to the wiki.*[30]

Both the HISP and the DHIS 2 wiki contains a news section on the front page where big meetings and conferences are announced.

5.4.1.2.1 Shortcomings of Confluence

I find that the Confluence wiki is lacking several useful and easy to use features that I am familiar with from other wiki software like MediaWiki and DokuWiki. The most noticeable shortcomings are that the entire page must be edited and not just a section, there is no separate discussion section (“talk page”) on each page, there is no category system and log messages and edit toolbars are absent. Confluence has similar functionality, like spaces instead of categories and comments instead of discussion pages, but they are not quite the same and do not fulfil all my wishes. Categories can be more finely divided than spaces, making it easier to find similar information about a given topic across spaces. I see the point of spaces, but would like to see categories as well.

I have not got the impression that anyone else I have spoken to within the HISP project has a strong opinion or is dissatisfied with Confluence. Whether this is because they have not used Confluence enough or if they find Confluence to be good enough, I am not sure.

When changing to Trac, it has been discussed on the mailing lists to move at least everything that is relevant for the DHIS 2 project from Confluence to Trac, which has its own wiki tool.

30 From the mailing list archive: http://www.hisp.info/archives/dev/msg02038.html

5.4.1.3 Javadoc

DHIS 2 has Javadoc in most Java classes. This is a randomly picked example from a Java class which contains the two attributes most commonly found in the Java classes in the DHIS 2 software, @author and @version:

```
/**
 * Defines the functionality for persisting DataElements and
 * DataElementGroups.
 *
 * @author [Developer 1]
 * @version $Id: DataElementStore.java 3419 2007-06-27
 * 16:04:27Z [username] $
 */
```

The Javadoc attribute called @author is where the name of the author is given. It has been a tradition to only list the developer who originally made the class, even if other developers have contributed with small changes afterwards. This is not formalized anywhere though.

The @version attribute is an automatically updated field that gives information about the version number of a class including when it was last edited, and who the author was.

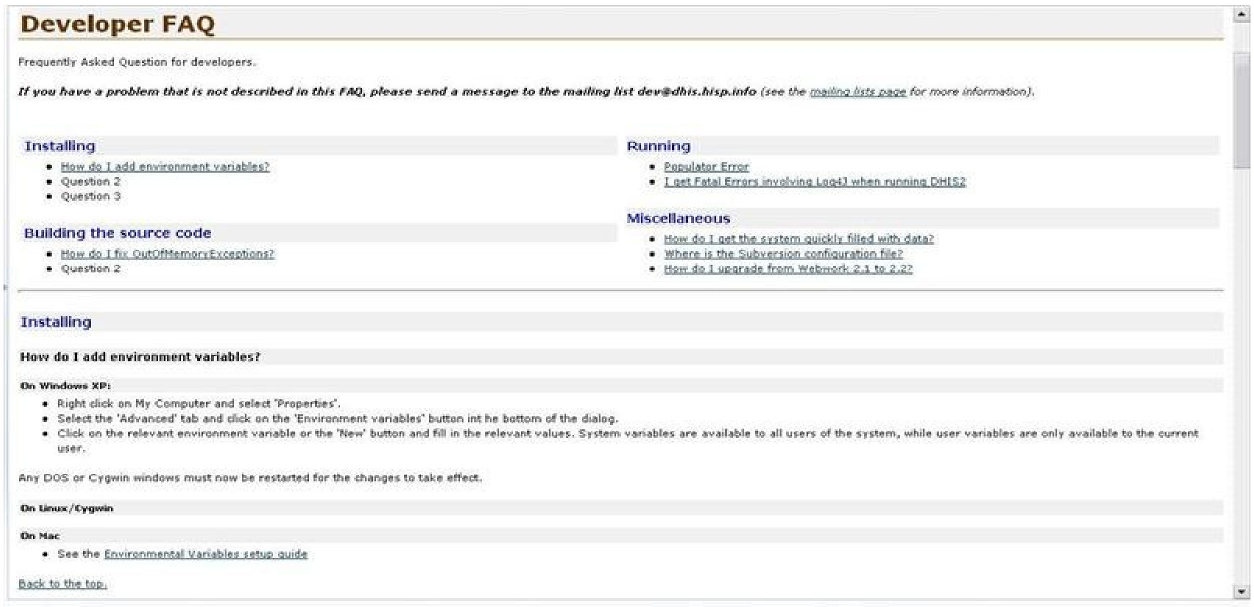
The Javadoc in DHIS 2 is published as part of the Maven-generated websites.

5.4.1.4 FAQ

From other software I have used or read about, I have found the FAQ a great place to start looking for answers when I want to know more about a given problem, get basic information or an overview of the software and its functionality. Since HISP had its own two FAQs I thought that would be a great place to start as well when I was new to the project. The only problem was that the first FAQ did not contain much information at all, while the second one contained a lot of outdated information. After I wrote an e-mail to the mailing list and consulted the other participants about the FAQ, we decided to delete one of the FAQs, and have three different FAQs; one developer FAQ for developers and problems they encounter, one administrator FAQ covering common issues in installation and configuration the software, and one planned FAQ for the users of the software. Later on, a system FAQ has been suggested as well, but it has not been written yet.

The FAQs are not frequently updated, and the administrator FAQ of DHIS 2 only contains 1 question, while the developer FAQ is a bit more popular with 7 questions (as of July 2007). There is no one responsible for converting questions and answers from the mailing lists into the FAQ, and there are no guidelines saying how this should be done.

Another problem with the FAQ was the layout. The FAQ consisted of answers and questions in random order and with no section containing an overview of all the topics. Since I was the one who made the new FAQs I decided, after receiving input from the dev-list, to have the questions grouped together in topics on the top of the page, and to use the question heading as linked text to the answer.



Developer FAQ

Frequently Asked Question for developers.

***If you have a problem that is not described in this FAQ, please send a message to the mailing list dev@dhis.hisp.info** (see the mailing lists page for more information).*

Installing

- How do I add environment variables?
- Question 2
- Question 3

Building the source code

- How do I fix OutOfMemoryExceptions?
- Question 2

Running

- Populator Error
- I get Fatal Errors involving Log4J when running DHIS2

Miscellaneous

- How do I get the system quickly filled with data?
- Where is the Subversion configuration file?
- How do I upgrade from Webwork 2.1 to 2.2?

Installing

How do I add environment variables?

On Windows XP:

- Right click on My Computer and select 'Properties'.
- Select the 'Advanced' tab and click on the 'Environment variables' button int he bottom of the dialog.
- Click on the relevant environment variable or the 'New' button and fill in the relevant values. System variables are available to all users of the system, while user variables are only available to the current user.

Any DOS or Cygwin windows must now be restarted for the changes to take effect.

On Linux/Cygwin

On Mac

- See the Environmental Variables setup guide

Back to the top.

Figure 8: Screenshot of the developer FAQ.

5.4.1.5 Documentation and recruiting developers

There has been participants in the HISP project who have chosen not to be part of the development team. One participant in the DHIS 2 project wrote to the mailing list, arguing that poor documentation, and especially poor source code documentation, was one of the reasons why he chose not to participate with any source code:

> *I'm one that chose not to code on dhis2, but rather work on other aspects of the project for several reasons, one of them being the experiences I've had with the code earlier.*[31]

5.4.1.6 Help functionality

Since I am not a user of the software, I do not personally need much help functionality, but it is still important to provide help to both new and established users of the software. Since DHIS 2 is being used in developing countries by people unfamiliar with computers, enough information has to be provided to make them understand how the software should be used.

The software supports several different languages, and internationalisation needs to be taken into consideration when help text are written. When we first installed DHIS 2 in the districts in Hue, we did not have any end user manuals. Since the users are not very familiar with computers, the lack of user manuals were noticeable. To compensate for this, we wrote down very basic instructions in text files for the users to read if they encountered any problems. Since these instructions have to be in Vietnamese, the Vietnamese participants are the only ones suitable of writing them. The HCMC team had written a user manual which the employee in Hue later customised for the users in Hue.

DHIS 2 does not provide much help functionality in the software at the moment, but how to provide it has been discussed several times both among developers and on the mailing lists. One of the student groups from Norway worked on integrated help in DHIS

31 From the mailing list archive: http://www.hisp.info/archives/dev/msg02407.html

2 in the fall of 2006.[32] They proposed several different kinds of help functionality in forms:

- CSS pop-up
- Classic pop-up
- In-line help

They reached the conclusion that in-line help, which is a short description of each important element, is not that efficient, since it takes too much space. CSS and classic pop-up have different pros and cons, and it has not been decided witch one is going to be used. A static manual as an independent document describing the overall concepts and structure of DHIS 2 has also been suggested.

Another question raised from the discussions was how the help system should evolve over time. When users get more experienced they need less help than new ones, and the help functionality should not get in the way of these users. It has not been decided how this is going to be solved yet.

DHIS 1.3 and DHIS 1.4, which has been around for a longer period of time, have better end user documentation and manuals. As for now DHIS 2 has some help documentation with various degree of content and length, in Gujarathi, Hindi, Malayalam, Telugu, Vietnamese, and English. These are available in the Beanish space on the wiki: http://www.hisp.info/confluence/display/BEANISH/BEANISH+documents.

5.4.2 Issue tracker

I have only used JIRA to a small extent, one of the reasons being that few others were using the tool.

In the fall of 2006 a student group was given the task of examining the project tools being used in the development of DHIS 2 and explore alternatives to these tools. One of the tasks was to set up Trac[33] and evaluate its usefulness versus JIRA, and especially how data can be moved from Confluence and JIRA to Trac, and how Trac integrates with other DHIS 2 tools.

Trac is an enhanced wiki and issue tracking system which uses a minimalistic approach to web-based software project management. The student group reached the conclusion: "*While Confluence may be more complex, Trac uses a seemingly cleaner and simpler design more inviting to the user. Code browsing and issue viewing/adding are easy to locate as they are central to developing software".*[34] Several people involved with DHIS 2 has argued that JIRA is too complex and that something simpler would be preferable. Trac might be the solution, but as the student group says: "*This does sound like a perfect fit for the HISP project, but ultimately it is the developers that must decide upon its usefulness*".

It was noticed in the beginning of April 2007 that only one issue had been registered in JIRA this year. That is less than what one would expect in an active, ongoing project. Another round of discussion was then conducted on the mailing list. One of the

32 The work is available at the wiki: http://www.hisp.info/confluence/display/RandD/Integrated+Help

33 Homepage of the Trac project: http://trac.edgewall.org/

34 The report is available online: http://www.hisp.info/confluence/download/attachments/10745/report.pdf

developers tried to take advantage of what the student group had archived last fall and began to set up a Trac.[35] Since Trac has been introduced, the project has seen an increasing use of the issue tracker, and over 100 issues have been registered so far. The issue tracker provides the possibility of assigning issues to users, but as of July 2007, only 11 out of 121 open issues has an owner.

Beside from being a place to register issues, Trac also serves as a roadmap showing which issues belong to which release, and which issues that have to be completed before a new release can be released.

5.5 Communication

5.5.1 IRC channel

An IRC channel was introduced to the HISP project during the summer of 2007. It has so far mainly been used by the Norwegian participants, but the Indians have also visited the channel to get support.

5.5.2 Shared knowledge

The mailing lists are one of the most central tools for collaboration in the development of DHIS 2. The dev-list is the far most used list for communication with about 375 topics discussed in the first six months of 2007. The HISP list is at second place, with about 40 topics in the same time span. It has been suggested to make several new mailing lists in addition to the ones already in use, one of them being an announcement list where new projects and releases are announced.

Information that is not going through the mailing lists is seldom shared between all participants in the DHIS 2 project. Some information is not necessary for all to know about, but I find the general information and knowledge sharing in DHIS 2 poor. I often get to know information, especially about meetings and visits to the other HISP nodes, accidentally when speaking to other developers or coordinators. The process of finishing a release was in the beginning coordinated between only a subset of the participants. This has been criticized on the mailing list, and one of the coordinators wrote the following during the release process of milestone 6:

> *Finally, as some of you know, and probably correctly have criticised, earlier I have coordinated this process of pushing and finalising releases through private communication. As you see I have changed my approach this time, so to those of you that have pushed a more open process, here it is, and now is your time to prove that it works better this way.... :)*[36]

After an initiative from a Norwegian developer situated in Vietnam for a period of time, the Vietnamese employees are writing their own individual weekly status update on the wiki to share what they are doing and which problems they are facing.[37] The Vietnamese developers have not always updated their reports on a weekly basis,

35 The DHIS 2's Trac page: http://www.hisp.info/dhis2

36 From the mailing list archive: http://www.hisp.info/archives/dev/msg03029.html

37 The weekly reports: http://www.hisp.info/confluence/display/HISP/Reports

although they are encouraged to do so. I started reading the weekly reports when I was situated in Vietnam, and I continued doing so when I got home. I seldom gave feedback to the employees based on what I read in their reports, but it kept me up-to-date on the situation in Vietnam.

The Hue team, which I was a part of, wrote our own weekly reports on the wiki during the time we were situated in Hue.[38] The amount of information given in this weekly update changed from week to week based on what we had done during the week and how much information we thought we needed to share. We only experienced getting feedback from the other participants based on what we wrote in our weekly report on a few occasions.

Since HISP is a global collaboration project, participants often attend conferences and meetings all over the world, but little of the information learned through these activities is shared on the mailing lists or on the wiki. HISP India has an event calendar on their web site showing upcoming events[39], but the calendar is infrequently used. Big meetings and conferences are announced on the wiki, but the rest of the meetings are not shared. A relevant topic is to share what was discussed at the meetings afterwards. Sometimes an e-mail is sent to one of the mailing lists, but most of the time, the project members not participating in the gathering are left uninformed of the outcome of these activities.

There are a number of systems being developed in parallel by HISP people, and to get an overview of the projects, one of the coordinators in Norway made a list of the systems on the HISP wiki early this year.[40] The list contains information about the technologies being used in each project, what the main development node is, and, if it exists, an URL to the project. The list is supposed to get updated by the different nodes when new projects start, and this has so far been followed up.

One developer in Norway made his own attempt to share information by writing on a private blog when he travels to different HISP nodes.[41] The first post was written when he attended a HISP conference in South Africa in the fall of 2006. Several posts were being posted during his stay in South Africa, and the blog was brought back to life when he visited India in 2007. These blog posts are not strictly HISP or DHIS 2 related, but conveys his thoughts and experiences with the project, the country, the people involved at the different nodes and the software.

Lack of knowledge sharing can result in work being done over again or the same topic being discussed several times between different project members. This is one example from the mailing list where a few participants were discussing writing privileges to the source code, when a third participant interposed and relates that he has already changed this:

38 The Hue weekly report: http://www.hisp.info/confluence/display/HISP/Hue+weekly+report

39 The event calendar on the HISP India web site: http://www.hispindia.org/index.php?action=viewmonth&module=calendarmodule&src=%40random451eb80353064

40 Overview of HISP projects http://www.hisp.info/confluence/display/HISP/HISP+projects

41 hansst's hjørne, the blog written by a Norwegian developer: http://hansst.blogspot.com/

> *I added *=r (read only access to everyone) to svnaccess a week ago or so. Sorry for not telling anyone. [Coordinator 1] knew, at least :) It's not documented on the wiki either.*[42]

In late May 2007 there was an initiative from one HISP participant to create a newsletter to share events and news from the different HISP nodes. It was proposed that the coordinators (or others appointed to do the task) from each country send a small note every month to one particular participant who puts it all together. It is said that the newsletter should contain information about new implementations, conducted training programs, new theses about HISP, new projects, software development activities or other relevant information. This newsletter has not been distributed yet.

42 From the mailing list archive: http://www.hisp.info/pipermail/dhis-dev/2007-May/000523.html

6 Discussion

In this chapter I discuss my empirical findings by drawing on literature and concepts presented in chapter 2 and 3.

6.1 Communication

The mailing lists are the number one form of communication in the development of DHIS 2. Creating the proposed announcement list will keep people who are only interested in information about releases and other announces updated on the current status without having to spend time going through all the other mails on the other mailing lists. This approach goes well with the argumentation of Holz et al. (1998) that sending information to people who are not interested in it, is as bad as having people not being informed on topics important to them.

On the other hand, increasing the number of mailing lists might not be such a good idea. Erenkrantz and Taylor (2003) conclude in their article that limiting the scope of discussion lists makes it easier for participants to understand and keep a record of what is going on in the project. They also argue that this must be balanced with the number of mailing lists and they see that having to many lists can be a problem as well. When the right numbers of lists is achieved, it allows people to easily classify and separate discussions based upon agreed topical lines (ibid).

It seems that the HISP and DHIS 2 projects have found the right balance of mailing lists for the moment, apart from the user list. The user list is hardly ever used, with only 5 e-mails within 2 topics being sent this year. The dev- and hisp-lists seems to have enough traffic to be useful and worth keeping. The despots-list also has enough traffic and important topics to be useful.

Erenkrantz and Taylor (2003) argue that asynchronous communication mechanisms, like e-mail, are usually preferred. This is the case in the DHIS 2 project, where most of the communication about the development goes on the public mailing lists. When synchronous meetings are required, they are usually held via instant messaging clients and the time are discussed up front and decided based on when the most people are available. A summary of the meetings is often presented on the developer's mailing list or at the wiki. This goes well with Erenkrantz and Taylor's statement that it is essential to make some form of archive of the communication when synchronous mechanisms are being used. Smaller synchronous meetings or IM-conversations between only a subset of the developers happens all the time, but logs or summaries from these meetings are seldom passed on to the rest of the project members. To keep everyone informed of what is happening in the project, summaries from these meetings or conversations should be shared as well. Knowledge sharing is further discussed in chapter 6.1.2.

6.1.1 IRC channel

HISP recently introduced an IRC channel to the project. Fogel (2005, p. 59-60) state that although it is possible to archive everything from an IRC channel, it is not necessarily the best thing to do. Many think of IRC conversations as informal, semi-private conversations, and do not want them preserved forever in an online archive (ibid). Nothing from the HISP channel has been saved and published, although Fogel argues that excerpts sometimes should be preserved. The HISP channel is a new initiative, which may explain why nothing has been archived yet.

6.1.2 Shared knowledge

Erenkrantz and Taylor (2003) state that a common problem in a distributed software project is to understand what other participants are currently working on. The weekly reports from the Vietnamese developers clearly show what is going on in Vietnam, including in which health care facilities the software is being implemented, what the developers are working on, which problems they are facing and so on. The people interested in what is happening in Vietnam can read their posts, and the developers writing the weekly updates can get feedback and help based on what they are struggling with. They can and should of course write to the dev-list if they have problems or questions, but writing down all kinds of problems and issues on the wiki shows what is going on in Vietnam. The initiative with weekly reports could be extended to all participants in all HISP nodes to follow the advice of Goldman and Gabriel (2005) to let everyone involved know what is happening in the project.

To fulfill the idea of letting people know what is going on, scheduling information should be shared for conferences, meetings or other happenings, where the event is held and who is attending. Although the HISP India calendar is not frequently used, a similar attempt for the whole DHIS 2 project would be a good information sharing initiative. A calendar would not be hard to implement on the wiki, but participants must update the calendar when they attend a gathering or are visiting abroad.

A related topic is sharing the outcomes and discussions from these events with the rest of the project afterwards. Sometimes an e-mail is send to one of the mailing lists, but most of the time, the project members not participating in the gathering are left unknown of the outcome.

The personal blog written by a developer in Norway is a more unusual way of sharing information, and this kind of initiative brings a personal feeling to the project. It also provides readers with a glimpse of what other participants in different countries are doing. I have been to Vietnam, and know the setting and how they are working there, but I have only met one of the Indian participants face-to-face and know few details about the case in India. The same goes for most of the developers.

As stated in chapter 3.4.2.3, DHIS 2 has a kind of module owner, but since this is not formalized in any document it is hard for new developers to know who is responsible for each module. All the Java classes has the Javadoc attribute @author, but this only gives the name of the original author, and since only the Java classes have this information, it is not always clear who to contact if you have a question. The dev-list can be used to get this information and ask questions, but it requires that extra step of writing an e-mail and waiting for an answer.

Little information about what developers and other project members are currently working on is shared on the wiki. I got to know the participants in Norway, and later on in Vietnam, by meeting them face to face. By reading the mailing lists and talking directly to the participants I learned which modules or part of the code they were currently working on. Other new project members may not have this opportunity, and when it is not formalized in any document, it can be hard for them to get an overview of the current status. With Trac, is it possible to see who is responsible for each issue registered in the tracker, but since only a handful of the issues have an owner, it serves as a poor place to look for what developers are currently working on.

The DHIS 2 project lists the project members with contact information on the wiki, which according to Goldman and Gabriel (2005) should be presented on the web site to help novices learn who is who in the project. Goldman and Gabriel even suggest that pictures of the participants are included. Confluence has user profiles for every registered user of the wiki. These pages can be used to provide information about a user, including a picture, but few of the HISP members has used this opportunity to share information about themselves.

Fogel (2005) lists several beneficial side effects gained from public discussions, among them that it can help avoid discussing the same issues over and over again. If all discussions in the DHIS 2 project were public, situations like the one where some developers were discussing writing privileges to the source code when these privileges had already been changed, would be avoided.

The initiative to create a newsletter to share news and events from the different HISP nodes to all participants fits well with Goldman and Gabriel's (2005) argument that people who do not have enough time to follow the mailing list should be informed about current issues in a regularly scheduled newsletter. Since the newsletter has not been distributed yet, it is hard to tell if it will serve this role in the HISP network.

6.2 Web pages and documentation

Coordinator 1 says:
Hi
I wanted to test this i18n feature
do you have some documentation saying how it works
Coordinator 1 says:
I mean user documentation [...]
Developer 1 says:
nope this is open source

Although the answer in this instant messaging conversation[43] between a coordinator and a developer of DHIS 2 is a joke, it still reflects the tendency of neglecting the writing of documentation in open source projects. Documentation has been discussed a number of times on the DHIS 2 mailing list, and even with a dedicated documentation project, DHIS 2 still lacks documentation on several topics.

Yeates (2006) list several ways to improve documentation. The first one, requiring structured documentation along with submitted source code, has been discussed in the DHIS 2 development as well. This approach requires more from the developers, who have to write documentation in parallel with writing code, but at the same time, the project is guaranteed to get some documentation and the documentation will be up to date. The first step to realising this method of structured documentation is to promote the use of Javadoc. Section 6.2.5 further discusses the use of Javadoc.

43 The whole conversation is available for download from the wiki: http://www.hisp.info/confluence/pages/viewpage.action?pageId=11790

The next point on Yeates' list, make as much as possible of the project information accessible to search engines, has also been discussed on the dev-list of DHIS 2. The documentation available on the wiki is searchable within the wiki. It has been suggested to upgrade or change this search functionality to make it even easier and better to use.

HISP changed to a new mailing system this year, making the new mailing lists searchable. The search functionality has not been fully implemented yet, even though it was said that it would be available right away. The old archives are still accessible, but not searchable.

The third point on the list is to encourage new users to contribute documentation as their first contribution. Yeates argues that new users are ideal for writing documentation aimed at new users, but this is not the experience I have from the development of DHIS 2, at least not when it comes to low-level documentation. My experiences is that new users do not know enough about the system to write good documentation that can be useful to others. One approach could be to have the new users write documentation drafts, which can be used as framework for experienced participants to write better documentation. Another possibility is to let new participants write documentation that does not require a deep understanding of the system, like for instance documentation on how to set up the system and install and configure software needed to do further development.

The final point on Yeates' (2006) list is allocation of explicit resources to writing documentation. Successful OS projects with external funding often use some of those funds to employ technical writers to write documentation (ibid). DHIS 2 has a tight budget and its money is not likely to be used on technical writers, but the project has access to participants who choose not to develop the software itself, and these participants could be encouraged to write documentation to a larger extent then what is the case today.

Fogel (2005) argues that documentation is never really finished, and that may be one reason why people delay starting writing at all. This seems to fit well with the development of DHIS 2 where everyone agree that documentation is important, but the actual work of writing it is postponed over and over. To get the basic documentation written, Fogel (2005) suggests that the scope is limited in advance and argues it will not feel like an open-ended task that way.

Appointing a leader of the DHIS Documentation project to organise the documentation, keep track of what needs to be written, encourage participants to write documentation and promote the documentation project, could be one approach to get people more aware of documentation and write more documentation.

Spinuzzi (2002) points out that accuracy can be one difficulty that can arise from an open system documentation process. DHIS 2 has a small development team where this has never been a problem, and I do not expect any of the participants to write wrong information on purpose. Furthermore, only registered users are allowed to edit the wiki pages, and since there are only a few contributors, it is not hard to examine what they are writing.

Lethbridge et al. (2003) found that people more often update issue tracker and source code documentation, and ignored complex and time-consuming documentation. This fits with my experience from the DHIS 2 project where most Java classes have some

Javadoc, but few sentences on the wiki page explaining how it works. Since Trac was introduced, the issue tracker has also seen an increase in its use.

A observation done in a survey by Forward and Lethbridge (2002) shows that several small to medium-scale software projects had little or no software documentation. The individuals in these project said they believed in the importance of documentation, but timing, budget and scheduling constraints left little time and resources to write documentation. This also fits well with my experiences in the DHIS 2 project.

6.2.1 End user documentation

Fogel (2005) argue that if the project members feel they do not have enough time to write documentation, they should start by focusing on documentation for the end-users. Yeates (2006) goes as far as to suggest that the most important aspect of documentation is to listen to the end-users' question and problem. He states that the documentation should be improved by first answering the end-users' immediate questions, followed by stepping back to examine and address the underlying causes of the problem. One reason why the end user documentation of DHIS 2 has not been prioritised, is that DHIS 2 is a rapidly changing software, and the end user manuals would need to be updated on a regular basis to reflect these changes. This is not an impossible task, but it takes time and no one has stepped forward and taken the task.

Another issue is that few of the end users can understand and read English, which means the end users documentation has to be written or translated into the various languages. Consequently, the project is dependant on participants with these particular language skills, making the task of producing timely and good documentation more complex.

6.2.2 Language barriers

None of the participants of DHIS 2 have English as their first language, but all the developer documentation is written in English. This could be a reason why people do not write documentation, but my experience with the project shows that language is hardly the main cause of why documentation is not being written. More or less all communication in the project is done in English, and language barriers is not a huge problem in other areas, although the English skills vary between participants. Norwegians receive several years of English training at school, but this is not the case in India and Vietnam, resulting in varying English skills between participants. The Vietnamese DHIS 2 employees in HCMC voluntarily attended English classes after work several days a week to improve their English skills.

Participants are encouraged to write documentation even if they find it hard to write in English. If someone first write the basis of a topic, other can read the proofs of the text, and improve it. This is another area where wiki come in handy. The pages are easy to change, making it less necessary to write good documentation at first since other easy can edit and improve the text.

6.2.3 Keeping the documentation up-to-date

Erenkrantz and Taylor (2003) state that it is often a problem to keep the end user documentation synchronized with the current version of the source code. Since DHIS 2 does not have much end user documentation, this is a minor problem, but keeping the rest of the project documentation up-to-date is also a challenge. As mentioned earlier, one participant wrote to the dev-list arguing that the documentation was outdated and that no documentation is better than wrong documentation. This does not correspond to the findings from Forward and Lethbridge's (2002) survey that concluded that document content can be relevant even if it is not up to date or Fogel's (2005) statement that even incomplete, rudimentary documentation is better than nothing at all. Forward and Lethbridge further argue that keeping the documentation up-to-date is a good objective, and based on my experience, the DHIS 2 participants share this point of view. The problem is that it takes time to write documentation, and the documentation is thus not always up-to-date.

The pages in the documentation space are not said to be finished after they are written, which is a common procedure found in several wiki pages, including the Wikipedia project. My experience from Wikipedia, especially with the smaller Wikipedia projects, like the one written in Norwegian, shows that one participant could start a topic by only writing a couple of sentences. Someone else might see this, have some more information to contribute with, and the topic is gradually more fully covered. The DHIS documentation space follows the same philosophy of stepwise improvement, although with a smaller number of participants.

6.2.4 The HISP wiki

To avoid destructive inputs by "trolls", the HISP wiki only allows registered user to edit and create new pages. Only allowing registered users to edit the wiki may result in fewer contributors, and a real threat to wikis are that nobody wants to edit them (Aronsson, 2002). Aronsson further argues that an active core of at least five regular contributors are needed to keep a wiki alive. It is hard to tell how many active contributors the HISP wiki has, as the wiki is divided into several small spaces where some users only contributing to one or a few of those spaces, but few participants contribute beside minor changes to the wiki. As one coordinator wrote on the mailing list: "most HISP people have yet to contribute to the wiki." He also wrote that a wiki is supposed to be a living document where everyone helps to keep it updated. This fits with the findings from a survey presented by Goldman and Gabriel (2005) where over half of the people did not read the mailing lists, but relied solely on the website for news.

Most developers only need to update information about the module they are working on, and as long as there are a few people willing to do the rest of the work, the wiki somehow survives. Goldman and Gabriel (2005) argue that the wiki cannot just be created and left to its own devices. They further argue that providing well-written content will work as a template for people to follow and will result in a better wiki since contributors imitate whatever patterns they see in front of them. The renovation of the HISP wiki, where old material was deleted, new documentation was written and the pages were reorganised into a better structure, should, according Goldman and Gabriel's (2005) thoughts, lead to a better wiki, and all the participants I have spoken to, agree that it has.

As stated earlier, wikis often suffer from a lack of navigational principles, duplication of information and an inconsistent target audience. I found all this to be true with the HISP wiki. To try to overcome these problems, the new documentation space was divided into different sections for users, administrators and developers. This makes it easier to navigate and find the proper documentation for a given audience. In the DHIS 2 space, I have reorganised the pages, trying to put everything related to the development of DHIS 2 together, making it easier to find the right information there also.

6.2.5 Javadoc

Goetz (2002) states that the provided Javadoc of open source software ranges from non-existent to poor, but that developers will still use the Javadoc to learn the code and the facilities, since other forms of documentation is not presented. The Javadoc in the DHIS 2 software is not organized to meet this reality and requires a lot of improvement to be useful as documentation. One example from a randomly picked method in a Java class, DataElementStore.java, shows this:

```
/**
  * Adds a DataElement.
  *
  * @param dataElement the DataElement to add.
  * @return a generated unique id of the added DataElement.
  */
int addDataElement( DataElement dataElement );
```

This Javadoc comment only tells the most basic information about the method, and most of the information would be possible for a developer to guess (a method called addDataElement is likely to be used for adding a data element). The information that should have been provided, following the list given by Goetz (2002) in chapter 2.2.2.7, includes how the method deals with bad inputs and error conditions, how this is communicated back to the caller, the method's pre- or postconditions, side effects and so on.

Goetz (2002) goes as far as to suggest that good code with bad documentation should be considered bad code, since it is more or less impossible to reuse the code. At the moment the DHIS 2 project has more then enough to focus on, and the Javadoc is given a low priority due to this. Most developers write some Javadoc when they implement new functionality, but it is not always as thorough as it should be. When I tried to understand the old validation code, mentioned in chapter 5.3, improved Javadoc would have made it much easier for me to understand the code and perhaps be able to use the code for further development.

Goetz lists a positive side effect of writing good Javadoc - it becomes a sort of code review. The only form of code review in the development of DHIS 2 is when the developers are paying attention to the scm-list, so pushing for better Javadoc may have several positive effects on both the software and the development process as a whole.

The Javadoc in DHIS 2 is published as part of the Maven generated websites, but when the sites have not been updated in 5 months, they are not useful as documentation for anything but old code.

6.2.6 FAQ

Collins-Sussmann et al. (2006) argue that a bad FAQ sheet is one that is composed not of the questions people actually ask, but of the questions the FAQ's author wish people were asking. The FAQs in the DHIS 2 project are made up of actually asked questions, which can explain why there are few questions presented, but it is probably not the main cause. The DHIS 2 mailing lists receives questions all the time, but few of these questions and answers are moved into the FAQ. Participants in the DHIS 2 project may not answer the same questions over and over again on the mailing lists, which is usually one of the reasons why projects make FAQs, but it should still be a policy to write on the FAQ when a problem is solved. The fact that no one is responsible for converting questions and answers from the mailing lists into the FAQ, or the lack of guidelines telling how this should be done, seems from my experiences to be among the main reasons why the FAQs contain few topics. Another reason might be the unawareness of the FAQs. References to the FAQ are seldom made on the mailing lists, and people seems to forget that the FAQs exist; thus they are not updated.

As Collins-Sussmann et al. (2006) says:

> *Compiling a true FAQ sheet requires a sustained, organized effort: over the lifetime of the software, incoming questions must be tracked, responses monitored, and all gathered into a coherent, searchable whole that reflects the collective experience of users in the wild.*

The DHIS 2 project seems from my experience to lack this sustained and organised effort when it comes to compiling a FAQ. The first step to archive a good FAQ would be to start using it. Writing guidelines describing how the FAQ is intended to be used, who is responsible for converting questions and answers into the FAQ and so on seems like a place to start. Secondly, making references to the FAQ on the mailing lists, especially if the question has been asked before, would make people more aware of its existence. Fogel (2005) also says that a FAQ can be one of the best investments for a project when it comes to educational pay-off. On the other hand, the participants seldom answer the same question over and over again (although it has happened), and the benefits the project could gain by expending resources on the FAQ might not be worthwhile when the development is on a tight schedule.

6.2.7 Documentation and recruiting developers

Documentation is a problem in most open source projects, and to prevent frustration and the loss of potential participants, projects need to have good internal documentation for developers (Goldman and Gabriel, 2005).

> *I'm one that chose not to code on dhis2, but rather work on other aspects of the project for several reasons, one of them being the experiences I've had with the code earlier.*

This statement from the dev-list shows that poor documentation can result in fewer developers, which correlate with the statement of Goldman and Gabriel (2005) that poor or nonexistent documentation will lead to frustration among potential developers which might give up. Fogel (2005 p. 26) state that when people abandon a project they abandon early, and therefore it is the start-up documentation, like how to install and get started with the software, that is the most important. Goldman and Gabriel follow up on this and state that the easier it is to learn how to get started, the more developers will be attracted to the project.

The survey done by Lethbridge et al. (2003) showed that documentation is important both when learning and when working with a new software system. My experience from the DHIS 2 project back up this find, and e-mails from the dev-list show that other participants have the same impression.

7 Conclusion

In this chapter I will summarise my research and make some concluding remarks in relation to my research objectives.

Primary research objective

Explore the challenges of providing documentation in open source projects.

During the process of learning the tools, frameworks and source code in the DHIS 2 project and developing a new validation module, I found the current documentation to be outdated, messy or not present at all. Other participants have expressed the same opinion on the mailing list, and this thesis has shown how documentation can be neglected in a given open source project. This is not because the project members do not want documentation. Findings from this thesis show that even if all project members agree that documentation is important, the time and effort needed to provide good documentation in sufficient quantities is not necessarily provided, mainly due to limited resources. DHIS 2 gained a lot, in terms of new and updated documentation, by establishing a separate documentation project and a separate wiki space to which the documentation pages were moved. This effort could be expanded to include more people and thereby create more and improved documentation.

After Trac was introduced to the project, the use of the issue tracker has increased compared to the usage levels seen with the previous JIRA solution. This is probably mostly because the Trac installation has a functioning e-mail alert system, illustrating the importance of e-mail as a basic tool in open source projects.

Javadoc documentation is frequently used to learn about the source code in open source projects, since other forms of documentation are often neglected. For instance, my experience is that more wiki-situated documentation about the workings and structure of the DHIS 2 code would be helpful, but since this sort of documentation is lacking, I had to rely more on Javadoc. The Javadoc in the DHIS 2 software is not organized to meet this reality and requires a lot of improvement to be useful as documentation. Poorly written Javadoc also makes it more difficult to reuse the source code.

The mailing lists of DHIS 2 are frequently used, and the associated mailing list archive serves as a complement to the documentation and knowledge transfer process, but the mailing list archive should be searchable to make it easier to use. Also, questions raised on the mailing lists are not systematically transferred to the DHIS 2 FAQs, making the FAQs less useful than they should be.

Knowledge sharing within the project has its limitations and only some reports and summaries from events are shared on the mailing list or the wiki. Establishing weekly reports for all project members, not only the Vietnamese employees, could be one way to improve the information sharing. Other attempts include making an event calendar, a project-wide newsletter, or, as recently introduced, an IRC channel where all participants can talk together.

Secondary research objective

Investigate how lack of documentation affects new project members.

My findings from the DHIS 2 project show that poor or lacking documentation can have a negative impact on the number of participants in the project. One participants says he is not willing to develop the software, based on his previous experience with the code and its documentation. Other participants, including the author, found the documentation messy and outdated, making it harder to contribute.

DHIS 2 is hoping to gain new participants the same way most other OSS projects gain members, and not only through the HISP course at the University of Oslo. However, without good documentation it can be hard to attract new members. The easier it is to learn how to get started, the more developers will be attracted to and engaged in the project, and the start-up documentation is thus the most important part of the documentation.

8 List of acronyms

AR	Action Research
BD	Benevolent Dictator
CVS	Concurrent Versioning System
DHIS 1	District Health Information Software version 1
DHIS 2	District Health Information Software version 2
FAQ	Frequently Asked Questions
FSF	Free Software Foundation
HIS	Health Information System
HISP	Health Information Systems Programme
HTML	Hypertext Markup Language
IM	Instant Messaging
IS	Information Systems
IDE	Integrated Development Environment
OS	Open Source
OSI	Open Source Initiative
OSS	Open Source Software
RDP	Reconstruction and Development Program
SVN	Subversion
UiO	University of Oslo

9 References

Aronsson, L. 2002 “Operation of a Large Scale, General Purpose Wiki Website”, *Proceedings of the 6th International ICCC/IFIP Conference on Electronic Publishing*, pp. 27-37. Retrieved May 29, 2007 from http://aronsson.se/wikipaper.html

Avison, D., Lau F., Myers M. and Nielsen, P.A. 1999 “Action Research”, *Communications of the ACM*, vol. 42, no. 1.

Baskerville, R. L. 1999 "Investigation information systems with action research", *Communications of the Association for Information Systems*, vol. 2, article 19

Berg, E. 2007 “The challenges of implementing a health information system in Vietnam”, Master thesis, University of Oslo.

Bonaccorsi, A. and Rossi, C. 2003 "Why Open Source software can succeed", *Research Policy*, vol. 32, no. 7, pp. 1243-1258

Braa, J. and Hedberg, C. 2002 "The struggle for district-based health information systems in South Africa", *The Information Society*, vol. 18, no. 2, pp. 113-127

Braa J., Monteiro, E. and Sahay, S. 2004 “Networks of action: sustainable health information systems across developing countries”, *Mis Quarterly*, vol. 28, no. 3, pp. 337-362

Collins-Sussmann, B., Fitzpatrick, B. W. and Pilato, C. M. 2006 Foreword from “Version Control with Subversion”, book compiled from Revision 2147. Retrieved May 30, 2007 from http://svnbook.red-bean.com/en/1.2/index.html

Cubranic D. and Booth, K. S. 1999 “Coordinating Open-Source Software Development”, *Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pp. 61-68.

DiBona, C. 2005 "Open source and proprietary software development" in DiBona C., D. Cooper and M. Stone (eds), *Open Sources 2.0: The Continuing Evolution*, O'Reilly Media, Inc., pp. 21-36

Drummond, J. G. 2000 “Open Source Software and Documents: A Literature and Online Resource Review”. Retrieved July 19, 2007 from http://www.omar.org/opensource/litreview/

Edwards, K. 2000 “When Beggars Become Choosers”, *First Monday*, vol. 5, no. 10, Retrieved April 23, 2007 from http://firstmonday.org/issues/issue5_10/edwards/index.html

Erenkrantz, J. R. and Taylor, R. N. 2003 “Supporting Distributed and Decentralize Projects: Drawing Lessons from the Open Source Community", *Proceedings of 1st Workshop on Open Source in an Industrial Context*. Retrieved May 18, 2007 from http://citeseer.ist.psu.edu/erenkrantz03supporting.html

Fogel, K. 2005 "Producing Open Source Software: How to Run a Successful Free Software Project". Retrieved November 26, 2006 from http://producingoss.com/

Forward, A. and Lethbridge, T. C. 2002 “The Relevance of Software Documentation, Tools and Technologies: A Survey“, *Proceedings of the 2002 ACM symposium on Document engineering*, pp. 26-33.

Gacek, C. and Arief, B. 2004 "The Many Meanings of Open Source", *Software, IEEE*, vol. 21, no. 1, pp. 34-40.

Ghosh, R. A, Glott, R., Krieger, B. and Robles, G. 2002 "Free/Libre and Open Source Software: Survey and Study FLOSS. Part IV: Surve" Retrieved January 18, 2007 from http://www.infonomics.nl/FLOSS/report/

Glass R. L., 2005 "Standing in front of the open source steamroller" in Feller, J, B. Fitzgerald, S. A. Hissam and K. R. Lakhani (eds), *Perspectives on Free and Open Source Software*, The MIT Press, Cambridge/London, pp.81-92

Goetz B. 2002 “Java theory and practice: I have to document THAT?“. Retrieved May 15, 2007 from http://www-128.ibm.com/developerworks/java/library/j-jtp0821.html

Goldman, R. and Gabriel, R. P. 2005 "Innovation Happens Elsewhere: Open Source as Business Strategy", Retrieved April 12, 2007 from http://dreamsongs.com/IHE/IHE.html

Hars A. and Ou, S. 2001 "Working for Free? - Motivations of Participating in Open Source Projects", *Proceedings of the 34th Annual Hawaii International Conference on System Sciences.*

Holz H., Goldmann, S. and Maurer, F. 1998 “Working Group Report on Coordinating Distributed Software Development Projects”, *Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 69-73.

INI 2007 "Open Source Initiative: History of the OSI". Retrieved July 17, 2007 from http://opensource.org/history

Lakhani K. and Wolf, B. 2001 "The Boston Consulting Group/OSTG Hacker Survey", Retrieved February 10, 2007 from http://www.ostg.com/bcg/

Lerner, J. and Tirole, J. 2000 “The Simple Economics of Open Source”, *NBER Working Paper No. 7600*. Retrieved April 23, 2007, from http://www.nber.org/papers/w7600

Lethbridge, C. L., Singer, J., Forward, A. 2003 “How software engineers use documentation: the state of the practise”, *Software, IEEE*, vol. 20, no. 6, pp. 35-39. Retrieved July 2, 2007.

Madsen, F. H and Nürnberg, P. J. 2005 “Calliope: Supporting High-level Documentation of Open-Source Projects” *Proceedings of the 2005 symposia on Metainformatics*

Nordal, K. 2006 "The Challenge of Being Open - Building an Open Source Development Network", Master thesis, University of Oslo.

Mockus, A, Fielding, R. T. and Herbsleb, J. 2000 “A Case Study of Open Source Software Development: The Apache Server”, *22nd International Conference on Software Engineering (ICSE '00)*, pp. 263-272.

O'Brien, R. 2001 “An Overview of the Methodical Approach of Action Research”. Retrieved April 20, 2007 from http://www.web.ca/~robrien/papers/arfinal.html

Raymond, E. S. 1998 “The Cathedral and the Bazaar”, *First Monday*, vol. 3, no. 3. Retrieved May 14, 2007 from http://www.firstmonday.org/issues/issue3_3/raymond/

Shaw, V. 2005 "Health information system reform in South Africa: developing an essential data set", *Bulletin of the World Health Organization*, vol. 83, no. 8, pp. 632-639

Spinuzzi C. 2002 ”Documentation, Participatory Citizenship, and the Web: The Potential of Open Systems”, *Proceedings of the 20th annual international conference on Computer documentation*, pp. 194-199

Stafford, T. and Webb, M. 2006 “What Is a Wiki (and How to Use One for Your Projects)”. Retrieved July 22, 2007 from http://www.oreillynet.com/pub/a/network/2006/07/07/what-is-a-wiki.html

Wikipedia 2007a “Software documentation”. Retrieved July 16, 2007, from http://en.wikipedia.org/wiki/Software_documentation

Wikipedia 2007b “Instant messaging”. Retrieved July 28, 2007, from http://en.wikipedia.org/wiki/Instant_messaging

Yeates, S. 2006 “Documentation issues in open source”, *OSS Watch*. Retrieved May 28, 2007 from http://www.oss-watch.ac.uk/resources/documentation.xml

Øverland, L. 2006 “Global software development and local capacity building: A means for improving sustainability in information systems implementation”, Master thesis, University of Oslo.

Appendix A

Agreement between Health Service of Thua Thien-Hue province, and the HISP project

Background

The Health Information Systems Programme (HISP) is active in a number of countries, and is commited to facilitate the introduction of computer based information systems for reporting and analysis at the district level and below, and to enhance the use of public health caredata at all levels. The HISP project in Hue was initiated in November 2004 and since then the software and routines for computerization have been piloted in first two districts, and then since March 2006 in five of the totally nine districts.

The HISP project provides a flexible open source software package, the DHIS, to support reporting and analysis of health information. This software has been co-developed and customized to the Vietnamese context by the HISP Vietnam teams in HCMC and Hue.

The software being used is a previous version of the DHIS; the MS Access based DHIS 1.4 and the next step in the project should be to upgrade to the newly released version 2.0. Following a successful software upgrade process the plan is to extend the project's scope to include all 15 statistical reports (B1-B15), and to involve all nine districts over the next 1,5 years. To support this expansion, the HISP Vietnam project would like to strengthen the HISP team in Hue with more technical staff. This agreement outlines how this upgrade and the expansion process will take place and how responsibilities will be shared among the two collaborating parties.

Timeline and action plan

September 15 2006:
Finish upgrade to DHIS 2, including database, reports and pivot tables for analysis. This first deliverable will include five (B1, B9, B10, B11, B12) of the 15 reports of the statistical health information system.

This system will be implemented in the five pilot districts during September. The data from the remaining four districts will be reported using the traditional paper forms from district to province and registered electronically at the province level to ensure full provincial coverage of the data (B1, B9, B10, B11, and B12).

December 15 2006:
Extend the data scope to include electronic reporting of all 15 (B1-B15) reports from the five pilot districts.

April 2007:
Extend the geographical scope to include totally seven districts.

July 2007:
Extend the geographical scope to include all nine districts.

December 2007:
All districts should by then report all 15 reports (B1-B15) electronically to the province level.

Responsibilities

TT Hue

- The health service will assign persons in all districts, who will be responsible for data entry, report production and training locally. These persons will be given sufficient time for training in and management of the system.
- The health service will provide sufficiently powerful computer systems for the remaining 4 districts, including printers for the production of local reports.
- The health service is responsible for supplying working place for Norwegian members of HISP Team in Health Service of TT Hue.

HISP

- HISP will provide the DHIS 2.0 software customized to support the T.T. Hue Health Service and the Statistical Division's HIS (B1-B15).
- HISP will continue to support running project implementation costs with a monthly contribution of 1.500.000 VND until the end of 2007.
- HISP will continue to support the salary of our employee in Hue until the end of 2007. She provides technical support to the project.
- HISP will dedicate one of the HCMC-based developers to work for the project in Hue. He will be based in HCMC and support software development from there, but visit Hue when necessary.
- Two Norwegian developers will be based in Hue from August-November 2006 to support the DHIS 2.0 upgrade process.
- The support for computerization process in the five pilot districts, HISP will provide 1 new computer and 1 printer to Nam Dong district and 1 printer to Huong Thuy district.

Dr. Nguyen Dung Dr. Duong Dinh Cong Ola Hodne Titlestad

Appendix B

DHIS 2.0 development/customisation for Hue:

1. Migrate database (orgunits, org hierarchy, data elements)

- Then add datasets in 2.0 GUI
- Then add data element and orgunit groups in 2.0 GUI

Deadline:	Wednesday August 9
Responsible Developer:	KA
Support:	Eivind

2. Migrate data values

Deadline:	Wednesday August 9
Responsible Developer:	Anders
Support:	KA, Thuy

3. Migrate reports (B1, B9, B10, B11, B12) to dhis-web-reporttool

Deadlines:

Report B10	Friday 11 August
All 5 reports	September 15
Responsible Developer:	Quang, Lars H. KA

4. Set up pivot table for health service and for district-level

Deadline:	Friday 11 August
Responsible Developer:	Eivind

5. Develop install package for Hue

Deadline:	Friday August 11
Developer:	Quang
Support:	Duc, Thuy

6. Customized data entry module

Deadlines:

First sample report (B10)	September 1
Prototype of a generic tool	September 29
First release	November 30
Developer:	Margrethe
Support:	Torgeir

Implementation

Phase 1 – testing and prototyping in T.P. Hue district

Time:	August 7 – September 15
Responsible:	Quang, KA and Eivind

1. Install first release of software Monday August 14
- Minimum 1 report ready (B10)
- Pivot table with data from this year (1st, 2nd quarter imported from 1.4)

2. Provide initial on-site training
- Week Aug 14-18
- 3 visits (2 hours sessions)

3. Follow-up training
- Next 4 weeks, Aug 21- Sept 15
- 1-2 visits a week

4. Bug fixing and software improvements
- Based on user feedback (be active in retrieving feedback)

5. Seminar on health management and information use

Phase 2 – implement software in the health service and three more districts
Time: September 15 – December 14

1. Install software in the four new offices
Responsible: KA, Eivind

2. Training seminar for all 5 offices
Time: Beginning of October
- Invite users from all five offices
- 1 day training seminar
Responsible: KA, Eivind
Support: The whole team in Hue

3. On-site training in all 5 offices
- Minimum 1 visit every two weeks to all five offices
- Prioritise **quick** response to support requests from the users
Responsible: KA, Eivind

4. Bug fixing and software improvements
- Based on user feedback (be active in retrieving feedback)
Responsible: Eivind, Margrethe

Phase 3 – Scale up to include all 15 reports

December 15, 2006 – March 31, 2007

1. Migrate the remaining reports from 1.4 to 2.0

Deadline:	December 15
Responsible Developer:	Quang
Support:	KA, Lars H.

2. Develop BIRT report templates for district and health service

Deadline:	December 15
Responsible Developer:	KA
Support:	Quang, Lars H.

3. Extend pivot table templates to include data from new reports

Deadline:	December 15
Responsible Developer:	Quang
Support:	KA, Lars H.

4. On-site training in the use of the new reports

Time:	December 15, 2006 – January 31, 2007

- Minimum 1 visit to each office every week

Responsible:	KA

5. Follow-up training at each of the five offices

- Minimum 1 visit every 3 weeks to each office

Responsible:	KA

6. Seminar in health management and use of information

Time: February/March 2007

- 1 day seminar for all district managers and health service director/managers
- link this seminar to the HIS course in HCMC

Responsible:	Cong

Appendix C

The topics covered in the documentation space are[44]:

- DHIS 2
  - User documentation
    - User manual: Manual on how to use the DHIS 2 application.
    - User FAQ: Common issues users might encounter.
  - Administrator documentation
    - Installing DHIS 2
    - Configuring DHIS 2
    - Design a DHIS 2 report with Report tool and iReport
    - Creating a DHIS 2 pivot table
    - Administrator FAQ: Common issues in installation and configuration.
  - Developer documentation
    - Downloading the source code: Instructions on how to retrieve the source code from our repository.
    - Building the source code: Instructions on how to build the source code.
    - Development environment and tools: A description of recommended tools and frameworks for developing, including downloading and configuring them.
    - Tools: The tools used in the developing of DHIS 2.
      - Windows
      - Linux
      - Mac
    - Development standards and conventions: A description of which standards, principles and guidelines to stick to when developing.
    - Developer FAQ
  - System documentation
    - System history: A short introduction to the evolution of DHIS, the motivation for the software and the health domain.
    - System overview: A short description of what this system is intended to do, which services it provide, and so on.
    - Concepts and techniques: An introduction to concepts such as layering, MVC, services, DAOs, inversion of control and the like.
    - Frameworks: The frameworks used in the DHIS 2 system such as Spring, Hibernate, Webwork, JUnit and the like.
    - The DHIS 2 data model: Description of the DHIS 2 model classes, such as DataElement, DataValue and the like.
    - Java API: JavaDoc of the DHIS 2 Java API.

44 From the Documentation space on the wiki: http://www.hisp.info/confluence/display/DOC/Home

        - Database API: Description of the database structure which lies under DHIS 2.
        - Modules
            - Overview and connections: A textual description of how the modules are connected, which depend on which and so on.
            - Import-Export Module
            - Data Provider Module
        - JavaDoc: External link to generated Javadoc.
        - System FAQ
    - Mailing lists: Mailing lists and rules for use of these.
- DHIS 1.4
- DHIS 1.3