University of Oslo
Department of Informatics

Open Source Software Development in Developing Countries

The HISP Case in Ethiopia

Nils Fredrik Gjerull
nilsfr@ifi.uio.no

Master thesis

September 15, 2006

Abstract

This thesis investigates free and open source software (FLOSS), and FLOSS in the context of developing countries. The research is based on two action research case studies. Both case studies are done within the Health Information Systems Programme (HISP) network. HISP is a research and development network focusing on promoting effective use of information in the health systems of developing countries.

The first case study was conducted in the Tigray region of Ethiopia. In this case study a team of researchers used action research to configure and adapt District Health Information Software (DHIS) to the local context of the Tigray health system. DHIS is a flexible health information system used to collect routine data from health systems. DHIS is distributed under a FLOSS license. I participated in this team as a software developer.

In the second case study I participated in the development of DHIS 2 which is a total reimplementation of DHIS based on a platform of FLOSS technologies. DHIS 2 is developed using distributed voluntary development and licensed under a FLOSS license. In other words DHIS 2 is developed using a community model commonly used by large scale FLOSS projects. I participated in this project as a FLOSS developer, and I focused on the extensibility of DHIS 2. I created a throwaway prototype of a plug-in framework.

Through this two case studies I investigate FLOSS and how FLOSS can benefit and are benefiting both Ethiopia and the HISP network. I argue that the access to source code facilitates technology transfer/translation of information and communication technologies (ICT). Context dependent software like health information systems need to be adapted to each local context in order to be useful and effective, having access to source code is a life saver in this process.

Acknowledgements

I give my warmest thanks to my family for their love and support. I thank my brother Ole Andreas Gjerull for encouragements, and for all the cups of coffee we have shared. I thank my parents Petter and Berit Gjerull for having faith in me and for economical support.

I thank Bjarne Holen and Henrik Aarseth, the two guys I share an apartment with. Thanks for all our conversations, and for frequently being the only social contact I had, after a long day of writing on my thesis. Thanks for all the hours we have shared in front of the television.

I thank Mina Lam and Thanh Phan for sharing dinners with me in the University canteen. Thanks for a lot of fun and interesting conversations, you have given light to my days. My master studies would not be the same without you.

I thank my research counselor Jørn Braa for constructive feedback and guidance. May HISP be successful in promoting the effective use of information in developing countries.

Warm thanks goes to all my colleagues in the Ethiopian HISP team. You have given a warm face to Ethiopia. I give special thanks to the other members of the Tigray team; Solomon Birhanu, Hirut Gebrekidan Damitew, Netsanet Haile Gebreyesus and Kalkidan Gezahegn. I thank all at the Tigray health bureau for making the Tigray project possible. I thank professor Sundeep Sahay for his assistance to us in Tigray.

I thank Knut Staring, Ola Hodne Titlestad and all the DHIS 2 developers for all their excellent effort in creating the next major version of DHIS. May simplicity and flexibility be your guides.

Table of Contents

Chapter 1

Introduction

Developing countries are historically disadvantaged, and the unprecedented rapid phase of technological development we see in our times, are leaving the disadvantaged in developing countries further and further behind. In the developed world the use of *Information and Communication Technology* (ICT) has skyrocketed in the last two decades. Most people in the developing world are left out of this development. The gab between those who can and those who cannot effectively use ICT are frequently labeled as the digital divide. This divide is not only a divide between those who have access and those who do not have access to technical gadgets. This divide is a knowledge divide which is not limited to the effective use of ICT, but also the production and development of ICT.

An important part of most ICTs is the software. Software are needed to make computers and mobile phones useful. The software in ICTs are frequently held secret by the producers of the software, and sold as products to customers. The customer is allowed to use the software, but not to change and distribute copies of it. The treatment of software as products is the most well known paradigm within the ICT industry. Another way to treat software has been visible from the early start of the computer industry. Here software is treated like literature open to peer review and editing. The literature I am talking about here is the source code. The source code explains in a human readable way what the software should do. The source code contains knowledge for those able and willing to read it. Access to the source code allows a user to change the software, and the user are also allowed to distribute copies of it. The software produced in this way is called free software or open source software, to incorporate both of this terms I will use the term *Free/Libre/Open Source Software* (FLOSS) in this thesis.

In this thesis I am going to explore FLOSS and how FLOSS can benefit the developing country Ethiopia. This I am going to do through two case studies conducted as part of the *Health Information Systems Programme* (HISP) network. I will base my research on a social informatics perspective on *Information Systems* (IS). And I will use action research as my research methodology. In the following sections I am going to give an introduction to this case studies, my motivations for doing this research, and the research domain and the objectives I had for the research. Last I am going to present the chapters in this thesis.

1.1 Two action research case studies

I have been actively involved in two projects. The first was an action research project conducted in the region of Tigray in Ethiopia. In the second project I participated as a developer in a FLOSS project. Both of this cases was conducted within the HISP network.

HISP is an international research and development network centered around health information. HISP have a vision to promote the effective use of information in the health systems of developing countries. Most of the people in this network are academics, health workers, and some ICT professionals. HISP are active in a number of counties in Africa and Asia, and in Norway where I am from.

In Ethiopia several regions had expressed interest in testing the health statistics software offered by HISP. The software, *District Health Information Software* (DHIS), is created to gather routine data from health systems. DHIS is designed to make it possible to customise it to the local context of various health systems, and it is licensed with a FLOSS license. Tigray was one of the regions to express interest in testing DHIS. In Tigray I was part of a team using action research to configure DHIS into the local context of Tigray. This was done within a social informatics perspective which consider both the social and the technical aspects of an IS.

After I had finished the Tigray case, I started on the second case study. A project had recently been initiated, with the mandate to develop a new version of the DHIS software called DHIS 2. DHIS 2 is a total reimplementation of the DHIS software. The development of DHIS 2 is done using distributed voluntary development, which is common for many large scale FLOSS projects. In this case I participated as a FLOSS developer, and I was working on a plug-in framework for DHIS 2.

1.2 Motivation

Through my studies in computer science I have been made aware of FLOSS. For years I have been interested in computers and in how computers works "under the hood". FLOSS has given me access to a lot of useful software which would otherwise be out of my reach. Having access to FLOSS has given me a great opportunity to learn about computers and programming. At the same time my Christian faith and my travels to counties poorer than mine, have given me an interest in developing countries. From this two interests it was not far for me to ask about FLOSS, and how FLOSS can benefit developing countries. By conducting my research as part of the HISP network I got an opportunity to conduct my research into FLOSS and developing countries, and at the same time I could actively contribute and become a better programmer.

Research into FLOSS and how FLOSS can benefit developing countries have caught relatively widespread interest in the last few years. I am not the first to ask this question. By basing my research on two concrete case studies I believe this research will create knowledge about FLOSS in the context of Ethiopia and in the context of HISP.

1.3 Research domains and objectives

The research domains in which I am going to conduct my research is best explained by Figure 1.1. A full line signifies that the relationship between the two domains is subject to my research. The dashed lines signifies relevant relationships which is not explicitly subject to my research, but they are relevant for my research. My problem domains are FLOSS, HISP, Ethiopia and the Tigray health bureau. I am going to concentrate on the relationship between FLOSS and Ethiopia, FLOSS and HISP, and between HISP and the Tigray health bureau. My research will be conducted using the framework of structuration theory and a social informatics perspective on technology. This will all be explained in chapter 2.

I have three primary objectives for my research into the previously mentioned research domains. This objectives will be presented in the following paragraphs.

Research objective 1 – *Explore the structuration of FLOSS.*
Through this research I want to learn about the structuration of FLOSS and

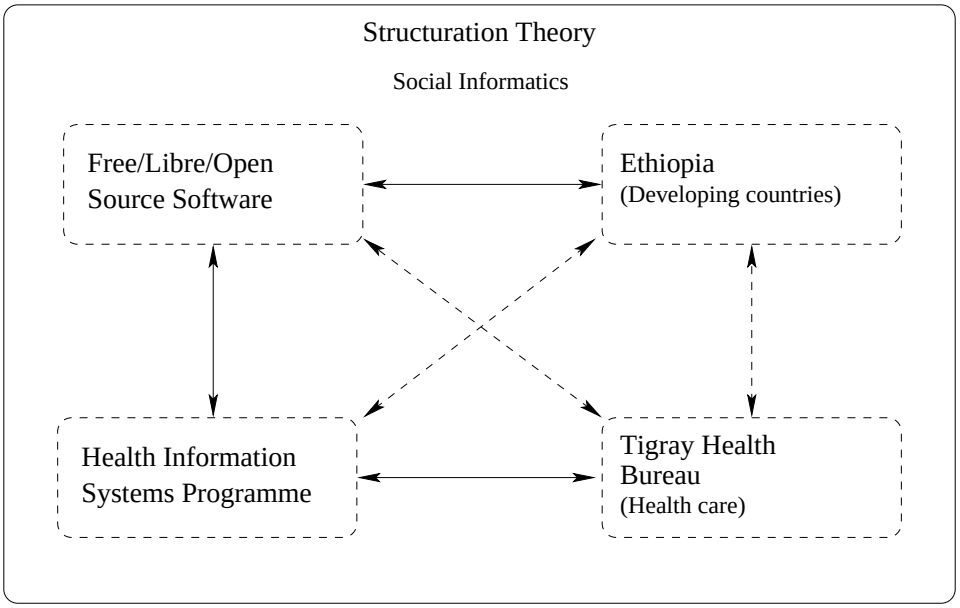

Figure 1.1: Visualising my research domains

common social practices of the FLOSS community. I will give special emphasis to the practices of source code sharing and distributed voluntary development. My exploration of the structuration of FLOSS will serve as an introduction to FLOSS for those who are not familiar with the FLOSS community, and will serve as background information for my case studies.

Research objective 2 – *Explore how Ethiopia can benefit and are benefiting from FLOSS.*
I want to explore the current use of FLOSS in Ethiopia, and equally important I want to explore the participation of Ethiopia in the global FLOSS community and the institutional support for FLOSS in Ethiopia. Since Ethiopia is a developing country I will also do research into how FLOSS can benefit and are benefiting developing countries in general. This research will be based on the concept of knowledge translation and how FLOSS facilitates knowledge translation. The transfer or translation of knowledge and technology, is important for bridging the digital divide. Based on the findings from this research objective and from research objective 1 I will discuss how Ethiopia better can benefit from FLOSS.

Research objective 3 – *Explore how HISP can benefit and are benefiting from FLOSS.*
HISP develops, support and distribute/translate the FLOSS licensed DHIS software. Through my research I want to find out why the DHIS software was licensed in this way and how this benefits the translation process, which DHIS undergoes in each local context where DHIS is used. My work in Tigray was primarily to translate DHIS into the local context of the Tigray health system. I also want to explore the development of DHIS and the differences between the development of DHIS 1 and DHIS 2. I want to compare the social practices in the current DHIS 2 development with social practices common in the FLOSS community. Based on the findings from this research objective and from research objective 1 I will discuss how HISP better can benefit from FLOSS.

1.4 Chapter presentation

Part I - Theory

Chapter 2 – Information System Theory: This chapter is a general introduction to information systems theories relevant to my research. This introduction is based on reviewing literature from the field of IS research. I start out by discussing different perspectives on IS and explain my chosen perspective. I explain the Scandinavian approach to systems design called participatory design, which is important to the design of DHIS. Last I review literature about IS development in developing countries.

Part II - Method

Chapter 3 – Methods: This chapter explains the methodologies and methods I have used in my research. I give a quite thorough explanation of action research. Action research is my overarching research methodology. The role I played in the two case studies are explained, and the time span and general context of the case studies are explained. I explain the concrete methods I use for my research. Last I make an assessment of the limitations of my research approach.

Part III - Background

Chapter 4 – Short History of Open Source: To explore the structuration of FLOSS I will give an overview of important movements, organisations and events in the history off FLOSS. I will start from the early history of computers and explain how the practice of sharing source code have replicated through time and space into our resent times. This is important for the understanding of the FLOSS community.

Chapter 5 – FLOSS - How does it work?: In this chapter I will explore the current social system of FLOSS. I will explain philosophies and values common in the FLOSS community. I will explain important development practices used by FLOSS developers and FLOSS projects. I will look into the question of motivation. What motivates people do use their time, energy and skill on contributing to FLOSS software, when they are not compensated for their time? I will look into governance structures within the FLOSS community, and I will look into how FLOSS projects are managed. Both proprietary software and FLOSS software come under the copyright regime, the difference is in the license under which the software is distributed. I will present the various types of FLOSS licenses. Then I will make an assessment of the challenges I see in the FLOSS approach to the development and distribution of software. Last I will explore FLOSS in the context of developing countries.

Chapter 6 – Health Information Systems Programme: This chapter introduces the HISP research and development network. The historical background to HISP is given and the basic philosophies of HISP are explained. I explains the most important methods and processes promoted by HISP, and how this methods and processes are inscribed into the DHIS software.

Part IV - Empirical Study

Chapter 7 – The Ethiopian Context: This chapter introduces the Ethiopian context. I give some basic demographics about the country, and I make a brief outline of the Ethiopian history and the current political system. Last I will explore ICT in Ethiopia. I will look into the current Internet infrastructure and the political plans to improve this infrastructure. The Ethiopian government has grand plans for the use of ICT in Ethiopia, this plans will be explained in this chapter. This chapter is meant to give an overview of the current state of affairs in Ethiopia.

Chapter 8 – The HISP project in Ethiopia: This chapter is based on my case study in Tigray. I introducing the Tigray context by giving some basic demographics about Tigray. I move on by introducing the team I worked with in Tigray, and our dealings with the Tigray health bureau. The Tigray health bureau already used a health statistical system so I will explain this system. Then I will describe our efforts to configure DHIS for the local context in Tigray, and the challenges we met in doing this. I will explain our relations with the rest of the HISP network. Last I will describe the cultural challenges I met while working in Ethiopia.

Chapter 9 – FLOSS in Ethiopia: I start this chapter by making an economic argument for Ethiopia to use FLOSS software. This is related to the TRIPS agreement which requires the signers to ratify laws protecting intellectual property rights. Then I look into the political support of FLOSS in Ethiopia, and move on to the institutional support given by FLOSS organisations. Then I look into the participation of Ethiopia in the global FLOSS community. Last I make an assessment of the use of FLOSS by analysing the software used on web servers in Ethiopia.

Chapter 10 – Development of a Plug-in Framework for DHIS 2: This chapter is based on the DHIS 2 case study. First I explain the reasons why a reimplementation of DHIS was necessary. Then I explain how the DHIS 2 development is organised according to a community model. After that I move on to explain my role in the DHIS 2 project, my motivations to participate and my approach for making a plug-in framework for DHIS 2. Last I explain my communication with the other DHIS 2 developers, and my communication with other FLOSS project I depended on for creating a throwaway prototype of a plug-in framework.

Part V - Discussion and Conclusion

Chapter 11 – Discussion: In this chapter I discuss the findings from my research into FLOSS, HISP, Ethiopia and developing countries. I discuss FLOSS related to Ethiopia and I discuss FLOSS related to HISP. Last I discuss more general theoretical considerations.

Chapter 12 – Conclusion: In this chapter I make an assessment of the validity of my research. I write some concluding remarks and suggest some avenues for future research.

Part I

Theory

Chapter 2

Information Systems Theory

Computers have their roots in the natural sciences, and was developed as a machine to compute numerical mathematics. Computers soon became more than number crunchers or calculators. They became useful instruments for collecting data, make calculations and present the data to the user. The computer became an information processing system, which gave the user more accessible and useful information. Many researchers have since theorised about the use, functioning and construction of computer based information systems. There existed information systems before the computer, but they were not labeled as such. An information system can potentially be completely paper based. I will focus on computer based ISs, but I recognise that computers are only one piece in a broader context. It is this broader context and the interplay of humans and computers that IS theory seeks to conceptualise.

In this chapter I will present the theoretical background relevant to my research. I will discuss different perspectives used to understand an describe ISs. I will start with a broad perspective and narrow down to my application area. After that I will describe the participatory design strategy for IS development. Last I will look into perspectives and strategies relevant for IS development in developing countries.

2.1 Perspectives

In this section I will start with a broad theoretical view and at a fast phase narrow down to the application domain of my research.

Computers have their origin within the confines of the natural sciences and its associated theoretical paradigm and methods. This has tended to give early IS research a technological focus and a positivist research tradition (Rose 2001). Positivism is a philosophy developed by Auguste Comte in the beginning of the 19th century, which stated that the only authentic knowledge is scientific knowledge. As a philosophy on science, it maintain that true knowledge can only be obtained from the data of sense experience. The validity of metaphysical speculation is denied. In positivism the world and the universe is seen as deterministic – they operated by laws of cause and effect. This causes and effects can be discerned if we apply the unique approach of the scientific method. The scientific method is believed to be an objective, value free way of viewing the world. Positivism believe in the idea that observation and measurement is at the core of the scientific endeavor. The experiment is the key approach of positivism to obtain observations and measurements under controlled conditions.

Positivism is to day rarely believed in the "pure" form described in the previous paragraph. In papers relating to IS it is frequently used as a label to describe a paradigm and methods more akin to the natural sciences with its belief in the hypothetico-deductive method and quantifiable data. Gregor (2005) even claim that positivism is not a defensible position in IS.

> The author believes that 'positivism' should no longer be even mentioned as a defensible position in discussions of theory or epistemology in information systems. If what is meant is a scientific perspective, then it is better to say so; to go directly to writings in the philosophy of science and to examine issues separately and carefully. The conclusion from this summary of positivism is that it is not a fruitful source of ideas on theorising in information systems.

(Gregor 2005, page 6)

As computers spread out of business head quarters and science labs, and into the society at large the theoretical paradigm and methods in IS have shifted towards those used in the social sciences. The use of computers have also grown into fields that are not characterised by mathematical rigor and predictability. Often IS seeks to capture information in contexts characterised by high degree of change and unpredictability. In other words the IS seeks to capture information about human society and not nature. This research paradigm in IS is frequently labeled as interpretivism or constructivism.

Interpretivism and constructivism are closely related to one another and

they reject the existence of theory neutral observations and the idea of universal laws as those in the natural sciences. This paradigm tend to have a preference of hermeneutic methods and qualitative data. Theory in this perspectives is not "discovered" as in "Newton discovered the theory of gravity", but it is constructed.

> Knowledge consists of those constructions about which there is a relative consensus (or at least some movement towards consensus) among those competent (and in the case of more arcane material, trusted) to interpret the substance of the construction. Multiple 'knowledges' can coexist when equally competent (or trusted) interpreters disagree.

(Denzin and Lincoln 1994)

IS relates to the natural sciences because computers are artifacts constructed with the use of physics, logic and mathematics, and it relates to the social sciences because it is an artifact used by humans for information processing and communication. A perhaps more explanatory and accurate label to use for this two different perspectives is the *natural science perspective* and the *social science perspective*. Which one of this perspectives that is most appropriate depends on what you are researching. If the subject of your research is characterised by high degree of control, predictability and do not have a high degree of human involvement, then perhaps a natural science perspective is most appropriate. This is not true for my research so I opt for a perspective closer to the social science perspective.

Having decided on a predominantly *social science perspective* we have to take what theory is most relevant for our study from the body of theory concerning IS. Given the high degree of human involvement in the subjects I am investigating a social systems perspective seems appropriate. A technology deterministic perspective, where the introduction of technology is seen as having a predefined effect on the organisation or community it is introduced to, gives technology a too elevated position. Technology is shaped by a community and will influence the community where it is introduced in one way or the other. The technology will again be reshaped by the community where it is introduced. Social science theory like Anthony Giddens' *Structuration Theory* (ST) (Giddens 1984) is held to be a useful theory for analysis of the change process ignited by the introduction of information systems into organisations and communities.

In the following subsections I will give a review of literature relating to the chosen social systems perspective and ST. In the last section I will go

into the application domain of my research. The introduction and development of technology take place within a context and in my case I have three contexts. One is the open source community and the second is the primary health care organisation of Tigray in Ethiopia. The third context of relevance is the HISP network and the community centered around the development of DHIS 2.

2.1.1 Information systems as social systems

Earlier models for measuring and predicting the social impact of technology characterised technology as tools, with a direct effect on the organisation where ICT is implemented. This line of reasoning is often focused on the distinctive features of a technical artifact, and imagines its use and effects based on this features. In many situations the impact of a technical artifact have diverged from the imagined effects. The Social Informatics Report (Kling et al. 2000) explains how introduction of new ICTs in several instances have failed to achieve it's goals, because the varied conditions under which people might use the ICT was not taken sufficiently into consideration.

> One key idea of social informatics research is that the 'social context' of ICT development and use plays a significant role in influencing the ways that people use information and technologies, and thus influences their consequences for work, organizations, and other social relationships. Social context does not refer to some abstracted 'cloud' that hovers above people and ICT; it refers to a specific matrix of social relationships. For example, social context may be characterized by particular incentive systems for organizing and sharing information at work.

(Kling et al. 2000, p.56-57)

Social informatics is an umbrella term for research which relates to the systematic research on the social aspects of ICT. This line of research point to the fact that direct effects arguments, that stems from a technological deterministic view on technology, have not generalized very well. Studies of the impact of ICTs usually show "mixed effects". The experienced "mixed effects" are in social informatics attributed to differing social and technical circumstances. It is insufficient to look at ICT's as technical systems. To be able to understand and predict the impact of an ICT both social and technical factors have to be taken into consideration. The social circumstances a new ICT is going to function in will most likely be non-trivial and the

effect of the ICT will most likely differ from what is predicted. Social informatics aspire to gather empirical data from both successful projects and from failed projects. This data is then used to explain and understand the reasons for the success or failures of ICT initiatives. This insights enables us to develop and improve our work practices.

Sawyer and Rosenbaum (2000) gives a summary of findings from social informatics research in seven points.

1. **The context of ICT use directly affects their meanings and roles.** Simply, context matters. The design of ICTs is linked to social and organizational dynamics, and these dynamics are contextual. This means that an ICT is always linked to its environment of use.
2. **ICTs are not value neutral; their use creates winners and losers** Given the contextual nature of ICTs, it follows that they are often designed, implicitly or explicitly, to support social and organizational structures.
3. **ICT use leads to multiple, and often paradoxical, effects.** The contextually-dependent nature of ICTs suggests that similar ICTs can have different outcomes in different situations. This also implies that ICT use can lead to both intended and unintended consequences. For example, new ICTs are introduced to one department in a local government to improve organizational effectiveness and efficiency. This leads to a state where that department staff's work processes soon become enmeshed with the new ICTs. The departmental staff becomes dependent on the infrastructure to do its work (the intended effect). However, the lack of systematic maintenance and upgrading of this infrastructure leads to the ICTs becoming unreliable. This lack of reliability means that, over time, the office is actually less capable of achieving its mission (an unintended effect).
4. **ICT use has moral and ethical aspects and these have social consequences.** The contextual nature of ICTs means that development and use raises moral and ethical issues. This set of topics often reflects the most well known of the key Social/Organizational Informatics issues.
5. **ICTs are Configurable – they are actually collections of distinct components.** The term, ICT, actually reflects collections of distinct components. These components – many of which are nearly commodities – are assembled into unique collections for each organization (or social unit, depending on the level of analysis). Furthermore, the multiple functions and ability to reprogram (or alter and extend) these functions makes any collection of ICTs highly re-configurable.
6. **ICTs follow trajectories and these trajectories favor the status quo.** The configurational ability of ICTs is underlain by the trajectories of the components. A trajectory means that any definable component can be seen as an evolving series of products (or versions). That is, they have a history and a future. And, the status quo means that preexisting relationships of power and social life are often maintained and strengthened. Since ICTs are socio-technical entities, their evolution is as much social history as technical progress.
7. **ICTs co-evolve during design/development/use (before and after implementation)** The configurational ability of ICTs also underscores the socio-technical process of ICT design, development and use is reflected in every stage of an ICTs life. A system's use unfolds over time in a form of mutual adaptation between the ICT

> and the social system into which it has been placed. This ever-unfolding process, a "design in use", also implies the variations in social power that define much of the discourse between ICT developers and ICT users.

The influence of technology on a community or organisation is not a one way street. As it is pointed out in the last three items above ICTs are configured and adapted to varying contexts. The same technical artifact will have different impact dependent on the social context, and the artifact will frequently be changed and sometimes be used in ways not imagined by the creators of the technical artifact. Williams and Edge (1996) elaborates on how technology is changed by the social context wherein it is applied:

> Whereas most contemporary applications of ICT have automated discrete, well-delimited functions, which can be standardised and readily obtained through the market, integrated applications of ICT to conduct a range of activities, can rarely be obtained in the form of standard solutions. Instead, firms must customise solutions to fit their particular structure, working methods and requirements. They may be forced to select, and link together, a variety of standard components from different suppliers. The result is a particular configuration - a complex array of standardised and customised automation elements. Moreover, no single supplier has the knowledge needed to design and install such complex configurational technologies. Instead, this knowledge is distributed amongst a range of suppliers (of different technological components) and a range of groups within the firm. Configurations are highly specific to the individual firms in which they are adopted.

2.1.2 Structuration theory in the field of IS

One of the principal aims of structuration theory (Giddens 1984) is to reconcile the two main strains in social theory. One strain of social theories place their focus on the individual, the human agents and human action, and how human agents and human action forms and remakes society. The second strain of social theories emphasises society, the structure of social systems, and how this structure enables or constrains human action (Walsham 1993). In this section I will investigate how ST can be applied in the IS field and specifically in my case study.

I will not dig deep into ST as this is not within the scope of this thesis. Rose (2001) have given an explanation of the facets of ST important to IS.

From this selection of facets I will select only those I deem relevant for my research. The selected facets I will describe in this section.

Agency

Human agency, in Giddens formulation, is the "capacity to make a difference" (Rose 2001). The capacity to make a difference is intimately connected with power. If you do not have this capacity you are effectively powerless. Power involves the exploitation of resources. There are two kinds of resources; authoritative resources and allocative resources. Authoritative resources is the power to coordinate the activity of human agents. Allocative resources is the power to control material products or aspects of the natural world. Power is not in it self a resource.

Much of the behaviour of human agents are subject to routines. Human action occurs as a "continuous flow of human conduct". This routines are replicated and changed over time and space. The human agent is consciously and unconsciously monitoring the routines he or she is following. Action have intended and unintended consequences. The human actor is continually altering the theories by which he or she is acting and doing sense making, in light of experience (Giddens 1984):

> All social actors, it can be properly be said, are social theorists, who alter their theories in light of experience – part of which experience is social theory. All theorists are likewise actors.

This is called the *double hermeneutic* by Giddens.

Structure

Giddens gives the following definition of structure (Giddens 1984):

> [Structure are] rules and resources recursively implicated in social reproduction; institutionalised features of social systems have structural properties in the sense that relationships are stabilised across time and space.

Structure exist only as memory traces and is instantiated, or made real, only in action. Social systems, which is reproduced social practises, do not have structures but have structural properties. The memory traces of structure orients the conduct of knowledgeable human agents. Structure can both be constraining and enabling.

The duality of structure

Having explained Giddens definition of agency and structure we now turn to one of the principal aims of ST, the reconciliation (or bridging) of the two. Giddens takes the, to some extend, opposing ides of structure and agency and recast it into a duality. Structure and agency are recast as two concepts that are depended on each other and are mutually changing each other over time and space. In Figure 2.1 social structure and human interaction are broken down into three dimensions, for the purpose of analysis.

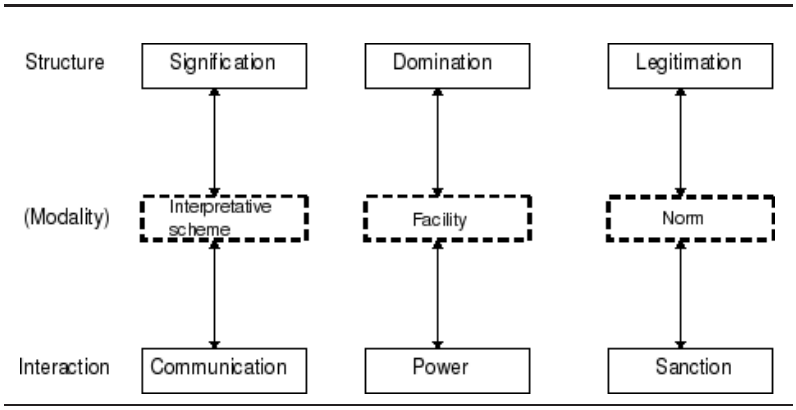

Figure 2.1: Dimensions of the duality of structures - Giddens 1984

Rose (2001) gives the following explanation to Figure 2.1:

> Thus, as human actors communicate, they draw on interpretative schemes to help make sense of interactions; at the same time those interactions reproduce and modify those interpretative schemes which are embedded in social structure as meaning or signification. Similarly the facility to allocate resources is enacted in the wielding of power, and produces and reproduces social structures of domination, and moral codes (norms) help determine what can be sanctioned in human interaction, which iteratively produce structures of legitimation.

Time space distanciation

The process of structuration is the evolution and reproduction of the duality of structure over time and spaces. The further the social practises embedded in a duality of structure is extended across time and space, the

better established they are, and the more likely to be thought of as an institutional feature of social life. People are aligned into a structure through social and system integration. Social integration refer to a relation of mutual dependence between agents, where the agents are all physically present. System integration refer to a relationship of mutual dependence between agents physically and/or temporally situated in different settings.

Critique of ST

An important critique of ST put forward by social theorists is the problem of reducing structure to action and consequently how to document an institution apart from action (Rose 2001). ST is also criticised for giving no direct answer to questions like "why do some forms of social reproduction succeed and become institutionalised, and others do no?". ST can help in understanding a current situation, but is criticised for not providing a conceptual base for developing a "critical" stance. ST give a picture of how things are, but not how it *should* be.

ST is a theory at a very high abstraction level. Consequently ST is criticised for been difficult to apply in any practical way to help in understanding the social context relevant for IS, and perhaps even help to predict consequences of an ICT. In order for ST to be useful in an applied science as IS should be, ST must be able to give understandings that can contribute to developing best practices and give insight to IS practitioners.

How to apply this in IS

Given ST's high level of abstraction, and given the high level of accuracy and find grained understanding required to make a computer based IS, it naturally follows that ST have to be simplified to be useful. ST has been used to theorise about ICT's and to analyse empirical situations (Rose and Scheepers 2001). As it is of fundamental importance to understand the context in which an ICT are going to be introduced, or whether an ICT should be introduced in the first place, it is helpful to be given a framework to aid in the analysis of a situation.

In his book Walsham (1993) present a framework for analysis which uses ST to connect the social context and the social process. The framework is lined out in Table 2.1. Here *content* is what the technological deterministic perspective focuses on, namely what an organisation "produces" and how they do it, and the software needed to make this process more effective.

Key Components of Change Framework	Associated Conceptual Elements
Content	Organisation – products/processes/systems Information systems – hardware/software/systems
Social Context	Web models – social relations/infrastructure/history Multilevel contexts
Social Process	Culture – subcultures/multiple meanings Politics – control and autonomy/morality
Context/process Linkage	Structuration theory – action and structure duality IS and modalities: • embody interpretive schemes • provide co-ordination and control facilities • encapsulates norms

Table 2.1: Walsham's analytical framework

The *social context* and the *social process* brings the social informatics perspective into the picture. By using the *duality of structure* Walsham brings the social context (structure) and the social process (agency) together. Walsham places IS in the modality realm that mediates between structure and agency.

Referring to the *time space distanciation* property of ST Rose and Scheepers (2001) uses Figure 2.2 to map the degree of "embeddednes" of social practises. As a social practice becomes relatively stable it will be a more likely candidate for an IS. An example of a social practice where an IS is used to structure the interaction is the practice of sending bug reports to an open source project. This practice is mediated by what is commonly called bug tracking systems, or issue management systems. When an IS is used to support a social practice it will most likely make the practice more structured, and less responsive to change.

Rose and Scheepers (2001) states that ICT is a powerful influence promoting time space distanciation. Discourse is the medium of structuration, it is through discourse the process of structuration is mediated. Interaction

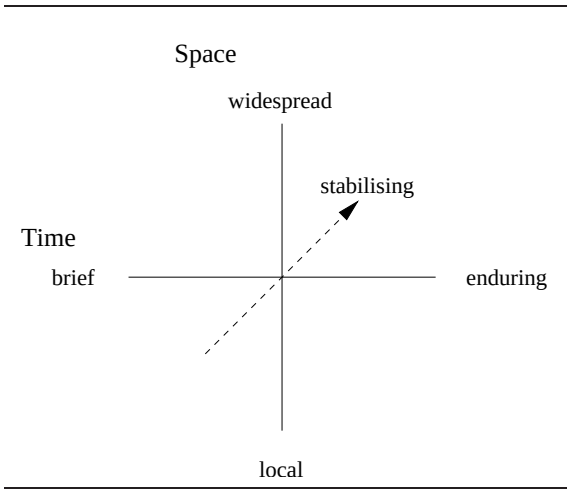

Figure 2.2: Social practices stabilising through time and space - Rose 2001

between agents are mediated by discourse, and the actors conception of the modalities that give a sense of structure are changed and replicated by discourse. Some of the more formal mediating roles of discourse can be supported by ICTs. Rose explains more on how IS relates to the *time space distanciation* of the *duality of structures*:

> [Information technology] does not embody structure. However, as a designed and managed artifact it is constituted, by human agents thorough a set of social practices involving IS professionals and others. ... As s product of human agency, ICT[1] will inevitably reflect the structures of the social system that designs and manages it, and their interpretation of the social system that it is intended to serve. Those interpretations, once embedded in silicon and software, may become relatively inflexible, compared to the development of social practices, and it is this inflexibility which is the source of the influence of ICT.

(Rose and Scheepers 2001, p.226):

2.1.3 Community- and Organisational Informatics

Having now spent some time in the abstract realm of social theory it is now time to get down to more specific theories relevant to my research.

[1] Originally labeled IT, but I changed it to ICT

My research domain is in three primary areas, the open source community, the HISP network and the primary health system of Tigray in Ethiopia. To get more insight into this three situation I deem it relevant with some basic theory from *Community Informatics* (CI) and *Organisational Informatics* (OI).

Organisational informatics

Organisational informatics is the older of the two. It was first during the nineties that computer came into widespread use. Before that time computers existed mainly within the confines of an organisation. So naturally research on ICT were also done within this confines. The focus was on the introduction of new ICTs and on how this influenced the organisation. Kling defines organisational informatics as follows:

> Organizational informatics refers to those social informatics analyses bounded within organizations, where the primary participants are located within a few identifiable organizations. Many studies of the roles of computerization in shaping work and organizational structures fit within organizational informatics.

(Kling et al. 2000, p.15)

Hanseth and Monteiro (1998) use the term *Information Infrastructure* (II) to denote the fact that computerised ISs have evolved into something more than the traditional monolithic system. Both within and outside the confines of an organisation ISs is made from a multitude of different technologies. II do not only refer to the technical components of a computer network, but it refers to the human and social components as well. When a new IS is introduced into an organisation there will most likely be an existing infrastructure that the new IS have to fit together with. The existing II in an organisation is called the *installed base* by Hanseth and Monteiro (1998). In order to give a definition of II they identify six aspects characteristic of IIs:

1. Infrastructures have a supporting or enabling function.
2. An infrastructure is shared by a larger community (or collection of users and user groups).
3. Infrastructures are open.
4. IIs are more than "pure" technology, they are rather socio-technical networks.

5. Infrastructures are connected and interrelated, constituting ecologies of networks.

6. Infrastructures develops through extending and improving the installed base.

In parallel with what we have seen in connection with the replication and changing of social systems in ST IIs evolves over time. If a social system have stabilised it will be difficult to change, and drastic changes will often be rejected. As we have seen, stabilised social practices are good candidates for ISs, and the ISs will serve to stabilise the social practice even more. This implies that the existing II, and by extension the existing social practices, will heavily influence how a new IS can be designed and how it will evolve. The successful introduction of new ISs are heavily dependent on how well the ISs fit with the installed base.

Community Informatics

The now widespread availability of computers in the developed world have brought the attention of researchers to phenomenon like *virtual communities*, communities that interact through the computer and that rarely or never meet face to face. In Giddens terminology this communities have strong system integration, but week social integration. It is my opinion however, that mediums like e-mail, *Internet Relay Chat* (IRC) and *Instant Messaging* (IM) diffuses the difference between social and system integration. This forms of interaction, though being physically and/or temporally situated in different settings, have a sense of more or less "face to face" interaction.

Community informatics is more than the research on virtual communities, it is the study of how ICT can be used to support more traditional communities. Wikipedia gives the following definition of CI[2]:

> Community Informatics, also known as community networking, electronic community networking, community-based technologies or community technology refers to an emerging set of principles and practices concerned with the use of Information and Communications Technologies (ICTs) for personal, social, cultural or economic development within communities; for enabling the achievement of collaboratively determined community goals; and for invigorating and empowering communities

[2]See: http://en.wikipedia.org/wiki/Community_informatics

> in relation to their larger social, economic, cultural and political environments. It can be considered as an socially-oriented and emergent sub-discipline of Informatics, itself a term with a wide variety of interpretations.

For my research it is more relevant to look at virtual communities as this is the kind of communities most common in the open source context. When it comes to developing countries, the use of ICT to empower local communities to reach collaboratively determined community goals is an interesting subject, and one that is being worked on. Gurstein (2003) proposes a community informatics strategy to reduce, or as he puts it, to go beyond the digital divide. I will review literature relating to the digital divide in section 2.3. Interesting as this is, it is not central to my thesis. My primary focus will be on *Virtual Community Informatics* (VCI).

Proulx and Latzko-Toth (2005) digs into the term "virtual community". This paper cites a traditional definition of community and gives the following explanation.

> [In the traditional definition of a community] we are confronted by a collective founded on geographical and emotional proximity, involving direct, concrete, and authentic interaction between its members.

(Proulx and Latzko-Toth) then goes on to investigate the meaning of the qualifier *virtual* to the word *community*. They retain three principal approaches to virtuality. In the first approach, the virtual is subordinated to the real. The virtual is a simulation of reality and therefor a false approximation. The second approach turns this on the head and says that "the virtual is to the real as the perfect is to the imperfect". The technologies of the virtual are perceived as liberatory. Global communication networks liberates human activity from the constraints of materiality, space and time. These two approaches rest upon a strict separation of real and virtual and are both imprinted with technological determinism.

The third approach recast the strict separation of the real and the virtual into a reality where the actual and the virtual is in a circular and productive relationship. This third approach works towards a "more textured understanding of the varying forms of virtuality worked through different technologies in different times and places".

The urban communities are in a sense "virtual" communities. The communities have emotional proximity, but do not necessarily have geographical

proximity. The people in a an urban community is bound together by common interest, like a church or a sport club, or bound together by friendship. These people can live all across the city. The people living in the same block and on the same street, which is a community in the traditional sense, can be emotionally far apart.

Unlike the traditional community bound by geography, like a village, where the commitment of the members necessarily have to be for a relatively long term, the commitment of the members in electronic collectives is generally more fluid. People are generally members of a multitude of communities, like a church, a family, a professional community, an academic community, a Linux user group etc. All these communities can have a level of virtuality, a Linux user group will be more virtual than a family. Virtual communities is like a desert watering-hole, where you meet, congregate and move on.

For most practical purposes I will focus on virtual communities mediated by ICT. I will use the definition of virtual communities proposed by Lee, Vogel, and Limayem (2002) as a working definition:

> [A virtual community is] a technology-supported cyberspace, centered upon communication and interaction of participants, resulting in a relationship being built up.

This definition is a synthesis of definitions used by various authors. A virtual community being a technology-supported cyberspace specify that a distinguishing feature of a virtual community compared to regular communities, is that a virtual community is mediated by computer networks.

2.2 Participatory design

According to Kling there is a significant body of organisational informatics research about the importance of user involvement in the design process. By involving users in the whole design process the hope is that the ICT will be more attuned to the needs of the people that are actually going to use it. System development based in the technological deterministic school was seen as a defined and controllable process with clearly defined phases. Kling argues that user feedback should be sought continuously.

> Most professional and educational literature still defines user involvement as assessing user requirements for a system on at

> the beginning of the design process. However, in these early assessments, many users emphasize the major functions and routines of their work, overlooking important variations or exceptions. If user feedback is not continuously sought throughout the design process, then a new system is likely to not effectively handle overlooked exceptions, complexities, and nuances.

(Kling et al. 2000, p.35)

Scandinavian research projects have traditionally put a strong emphasis on user involvement. In the Scandinavian research tradition user involvement has been discussed and practices in the last 30 years. This approach to design has been given the label *Participatory Design* (PD).

Bjerknes and Bratteteig (1995) list the three reasons normally given for user participation in design:

1. Improving the knowledge upon which systems are built.
2. Enabling people to develop realistic expectations, and reducing resistance to change.
3. Increasing workplace democracy by giving the members of an organisation the right to participate in decisions that are likely to affect their work.

The two first reasons are based on practical considerations, common to several system development approaches. The third reason is based on a belief in democracy and that it is a valuable goal to increase workplace democracy. To achieve this PD should involve future users in decisions taken during system development (Bjerknes and Bratteteig 1995). The third reason give PD a strong political touch and potentially makes it deeply controversial from a management perspective.

PD have it's background in the Scandinavian trade union projects in the early 70's. Within the Scandinavian IS research tradition the main theoretical contribution have come after projects, as researchers have reflected on the projects. The projects conducted after the trade union projects can be organised into two different trends; *the design for the skilled worker*, and *use of computers in an organisational context*.

The first trade union projects had some characteristics in common. This projects were mainly concerned with the organised work force and production. They claimed that there is a antagonistic relationship between labor

(the workers) and capital (the management), and wanted to strengthen the labor side in order to make the struggle more even. They claimed that researchers have a duty to support those with less power, and were striving for a democratic research and development approach. One of this project was initiated by the *Norwegian Iron and Metal Workers' Union* (NJMF) with the objective of applying a workers' perspective on development and introduction of new technology (Bjerknes and Bratteteig 1995).

Two projects represents the next generation of projects; UTOPIA which represents the *design for the skilled worked* trend, and Florence which represents the *use of computers in an organisational context* trend. Both trends saw it as necessary to create alternative technologies, to fight vendors' monopoly. The trade union projects did improve workers' influence on technology, but to increase this influence the focus was shifted to the form and content of the work process.

The UTOPIA project was conducted between 1981 and 1984 as a joint research project involving several Scandinavian research institutions and the Nordic Graphical Union. The project focused on work processes concerned with page layout and image processing in the newspaper industry. This project was laboratory based, where trad union representatives would participate as skilled workers. In the laboratory, mock-ups and simulations of computer based working environment was used. This approach called *design by doing*, allowed the graphical workers to express their craft skill by demonstrating their work. At the end of the project the *tool perspective* was developed:

> The computer should be a tool for the skilled worker, and the worker should be in control of the tool.

(Bjerknes and Bratteteig 1995, p.78)

The Florence project focused on the nursing profession and was conducted from 1984 to 1987. Essential to the project was *mutual learning*. This is a process were the users and the designers learn from each other, which is important to PD. The Florence project started out focusing on a single profession, but a strict bias towards the nursing profession was difficult to maintain. Physicians and nursing assistance was also involved. The project discussed the IS relating to the organisation as a whole.

> The totality of an organisation can be addressed in two ways, through a management perspective or by emphasising that there are several differing perspectives depending on various stakeholders' organisational positions and roles.

(Bjerknes and Bratteteig 1995, p.81)

It is important to note that an important aspect of user participation is the users' sense of ownership over the system.

> ...the users' sense of ownership is significant for the sustainability of the system. Lorenzi and Riley suggest that technically competent [ICT-based] systems may be woefully inadequate if their implementation is resisted by people who have low psychological ownership of that system. On the other hand, people with high ownership can make a technically mediocre system function fairly well.

(Nhampossa 2006, p.68)

In the European tradition of PD there are two methods; the *socio-technical* approach and the *collective resource* approach. The socio-technical approach considers the organisation as a whole and stresses that employers and employees have a common interest in developing useful ISs and therefor seeks consensus among the different stakeholders of the system. The collective resource approach emphases the conflict of interest between employers and employees. This approach highlights questions about power, control and democracy at the workplace and regards it as the researchers' responsibility to support the weaker party. (Bjerknes and Bratteteig 1995, p.83) and (Nhampossa 2006, p.69-71).

The US tradition of PD have downplayed *democratic* empowerment, that is to give workers decision making power, and focus more on *functional* empowerment, that is to empower workers to have more freedom in *how* to effectively meet the goals set by management. In the US, focus shifted towards approaches like *contextual design* which, while maintaining the importance of user involvement, gave the prerogative to the expert designer and emphasised management goals like efficiency, productivity and flexibility (Spinuzzi 2002). In the US, traditions like *user-centered design* and *customer-centered design* have emerged. This tradition do not focus on democracy and empowerment, but on making useful products for the customer or end users. Xerox *Palo Alto Research Center* (PARC) was an early supporter of PD in the US (Asaro 2000).

2.2.1 Prototyping

A very important method used to facilitate effective communication between users and developers is the use of prototypes. Both the UTOPIA and

the Florence project used prototyping as part of their strategy to facilitate user participation. In the Florence project prototyping reduced misunderstandings in the communication between the workers and the researchers and it helped in clarifying the needs of the nurses. The UTOPIA project used mock-ups as prototypes of a future system in order to make a requirement specification. In this section I will describe the two primary types of prototypes that are commonly identified; The *throwaway* prototype and the *evolutionary* prototype. Last I will present the different ways prototypes have been used in PD.

Throwaway prototyping is also called *rapid prototyping*. This class of prototypes is not meant to be a part of the finished system. It is only meant to give a visual illustration of the system, or parts of the system. This is done in order to facilitate communication between users and designers. The UTOPIAn mock-ups are an example of this class of prototypes.

> Rapid Prototyping involves creating a working model of various parts of the system at a very early stage, after a relatively short investigation. The method used in building it is usually quite informal, the most important factor being the speed with which the model is provided. The model then becomes the starting point from which users can re-examine their expectations and clarify their requirements. When this has been achieved, the prototype model is 'thrown away', and the system is formally developed based on the identified requirements.

(Crinnion 1991, p.17)

Throwaway prototypes can be further classified according to their fidelity, that is how close the prototype resembles the actual planned design. Examples of low fidelity prototypes are paper based prototypes. Using paper, scissors, pens, tape etc the user interface can be pasted together in collaboration with end-users. High fidelity prototypes are interactive and simulates parts of the planned design. A high fidelity prototype can be implemented using a tool for building a *Graphical User Interface* (GUI) which looks like the target system, but have now functionality, this is called horizontal prototype. The prototype can focus on only part of the functionality, which is called a vertical prototype (Rudd et al. 1996).

Evolutionary prototyping is a different way of thinking about prototypes. Here the prototype is meant to be a part of the target system. The prototype is coded with a quality level high enough for inclusion in the target system. Evolutionary prototyping is iterative and in each iteration only

well understood requirements are implemented. After the well understood requirements are implemented chances are that you will understand the other requirements better. The iteration goes on indefinitely. The system is put into use as soon as a minimum number of requirement are met, and is constantly refined based on experience from using the system (Davis 1992).

> For a system to be useful, it must evolve through use in its intended operational environment. A product is never 'done', it is always maturing as the usage environment changes . . . we often try to define a system using our most familiar frame of reference – where we are now. We make assumptions about the way business will be conducted and the technology base on which the business will be implemented. A plan is enacted to develop the capability, and, sooner or later, something resembling the envisioned system is delivered.

(Software Productivity Consortium 1997, p.6)

In the PD tradition the importance of prototypes to facilitate user - developer communication has long been understood. In Scandinavia Bødker and Grønbæk (1991) proposed *cooperative prototyping*. The prototyping process is seen as a cooperative activity involving users and designers, where the users and designer should be equally active drawing on their different skills. Cooperative prototyping focus on high fidelity throwaway prototypes, and is very design focused.

In the US two notable prototyping techniques were developed; *Plastic Interface for Collaborative Technology Initiatives through Video Exploration* (PICTIVE) and *Collaborative Analysis of Requirements and Design* (CARD). PICTIVE and card are both focused on building low fidelity throwaway prototypes and resembles game playing. (Spinuzzi 2002)

2.2.2 Limitations and challenges with participatory approaches

The value of participation have broad recognition in the IS literature, so much that Heeks (1999) gives it the status of orthodoxy. Like other methods that have received wide recognition PD has been in the danger zone of becoming a "silver bullet" method, a method that is always relevant and beneficial. In practice participation is often beset with problems. In this section I will present some of the problems articulated in the IS literature.

Inherent within PD are conceptions of democracy and empowerment. All stakeholders in an IS should be given influence on the design. This implies

that a team should include all stakeholders and that they must participate on equal terms. In contexts where there is a gap between this conceptions and the realities of the organisation, the ideal-reality gap can make participation unattainable or severely hampered.

Heeks, Mundy, and Salazar (1999) gives some examples of situations where user-participation techniques are unlikely to work well:

- Users lack information about participative techniques and about the new information system.
- The objectives of senior staff are not to share power and the values of the organisation are authoritarian and hierarchical.
- Users lack the skills and confidence necessary to engage in participative processes.
- The management style and organisational structures of the organisation are highly centralised.
- The organisation lacks the time and money to invest in participative approaches.

In his paper Heeks (1999) suggest three question that should be asked when participation are considered:

1. What is the political and cultural context?
2. Who wants to introduce participation, and why?
3. Who is participation sought from? Do they want to, and can they, participate?

By asking this questions factors limiting the value of participation can be revealed. Heeks (1999) explains situations that limits the value of participation and problems with participation; on who is chosen to participate, and the difference between participants in their ability to make a difference. I will not go through them here, but this points are important to be aware of when participation is considered.

2.3 Information Systems and developing countries

As already mentioned multiple times the context of an IS matters. As my research takes place within Ethiopia, which definitely is a developing country, it make sense to find some challenges commonly met in developing

counties when implementing an IS. Developing countries are very diverse, so I will have to paint with a broad brush here. Challenges meeting developing countries are many, but I will focus on challenges in connection with ICT and IS. In this section I will first explain the importance of social informatics and cultural understanding when working with ICT in developing countries, then I will focus on the worthy goals of increasing democracy and empowering the marginalised through ICT. After that I will review some literature about the *digital divide*. Last I will review literature about the transfer or translation of western technology into the different contexts in the developing world.

Walsham (2001) argues that social systems methodologies that emphasises the importance of the organisational, cultural, social and political context are highly suitable for developing countries. Culture are often portrayed as constraining, inhibiting the effective use of technology, by western analysts. This perception is marked by heavy cultural bias. This does not mean that we should naively accept all aspects of a culture, but we should think twice before labeling aspects of a culture as limiting. Perhaps the technology is inappropriate, not the culture. Most of the worlds ISs are made for western markets. From ST we can say that this ISs seeks to support social practices common in the western world.

Walsham emphasises the importance of obtaining deep understanding of the local culture when working with ICT in a particular context. A lot of understanding can be obtained by reading extensively about a particular region or country, but to really understand the subtleties of a culture and its social rules you have to immerse yourself in the culture.

> There are various way in which cultural understanding can be developed, not least by living in a particular country, and thus being immersed in the culture. ... An expatriate manager of a multinational company, staying at a five-star hotel, may be physically present in a particular country, but may have little access to or interest in local culture. Understanding through immersion require a starting point of respect for local cultural values, and considerable effort to understand these.

(Walsham 2001, p.201)

Braa (1997) argues why social systems methodologies are even more appropriate in the developing world than in the developed world. The social systems in the developing world tend to be more fleeting and informal. Stable structures are easier to formalise, and developing countries tend to have more unstable structures. As mentioned earlier, the more stable a structure

is the better candidate for an ICT it is. Within developing countries development can be at very different stages, with substantial differences between different regions and between urban and rural areas.

Braa argues that the Scandinavian participatory approach to system development can also be useful in developing countries. The Scandinavian approach focus on the local scale, process, empowerment and mutual learning. A typical scenario in developing countries "40 people, 20 units and 1 computer". This makes ISs into predominantly social systems, with some computerised support. With few computers and little ICT experience the need for support/training is important and will have to be established during the development process. IS development in developing countries need to be rooted in the local social system and driven from within. To attain sustainability a process which leads to empowerment and a sense of ownership towards the IS have to be cultivated.

Kimaro and Titlestad (2005) introduces the concept of *participatory customisation*. It is within the same tradition as PD, but shifts the focus from designing a system from scratch into adapting a preexisting system to a local context. Users, that are not computer savvy, should be able to make basic changes.

> Customisation means that the intended users change the system design in order to reflect their work practices and needs. The design of an already existing system is customised with user participation where intended users, not necessarily with high technological skills, are initially trained to be able to participate.

Because of limited resources in developing countries it makes sense to adapt an already existing system, rather than building from scratch. This approach have challenges similar to other participatory approaches, like motivating and selecting the right participants, but it is even more important that the participants develop basic computer skills. A customisable system should have the ability to easily implement visible changes.

2.3.1 The digital divide

The digital divide is the increasing gap between the people that do and the people that do not have access to computers and computer communication. This do not only refers to computers and computer networks, but also the knowledge needed to make use of computers. There is a digital divide between countries, between the developed and developing world.

Within countries there is a digital divide between urban and rural areas, this is especially evident in the developing world. There is also a digital divide between the different strata of society, like between educated and uneducated people. (Gurstein 2003)

If the digital divide is not bridged it is believed that marginalised groups might become even more marginalised. ICT gives those with the ability to effectively use the technology an advantage compared tho those who are not able to effectively use ICT. A popular label for the time we live in is the *information age* and ICT gives access to a vast body of information, through the Internet for the most part. The **C** in ICT is also important through technologies like the Internet and to a lesser extent through conventional communication technologies like a telephone (fixed or mobile), you can communicate with people all over the world. You can promote your views and explore information about subjects that interests you, or you can sell handicraft or buy a digital camera. In other words you can participate in an emerging virtual marked and a global virtual community. The hope is that the bridging of the digital divide will improve social and economic equality, academic advancement and self improvement, economic growth and democracy[3].

The arguments used to advocate the importance of bridging the digital divide are flavored by techno-optimism. Even if digital divide proponents states that it is not a panacea they non the less predicts that ICTs will have a substantial positive impact. Gurstein (2003) argues that the digital divide rhetoric to narrowly focuses on *access* to computers and the Internet, and go as far as to conclude that the concepts and strategies underlying the notion of the digital divide are little more than a marketing campaign for Internet service providers. In his article Gurstein argues for a shift from a narrow focus on access to a focus on what he labels *effective use*. Effective use he defines as follows:

> The capacity and opportunity to successfully integrate ICTs into the accomplishment of self or collaboratively identified goals.

Warschauer (2002) gives three case examples where efforts at improving peoples life through ICT had disappointing results due to the lack of consideration of the socio-technical context of the case sites. This three cases too narrowly focused on providing hardware and software. Warschauer have categorised the resources needed to make effective use of ICT into four categories illustrated in Figure 2.3. This resources have an iterative

[3]Have a look at http://www.digitaldivide.org/, http://www.digitaldivide.net/ and http://www.bridgethedigitaldivide.com/ for more information about the digital divide.

relationship with ICT which can lead to an upward or downward spiral to the effective use of ICT.

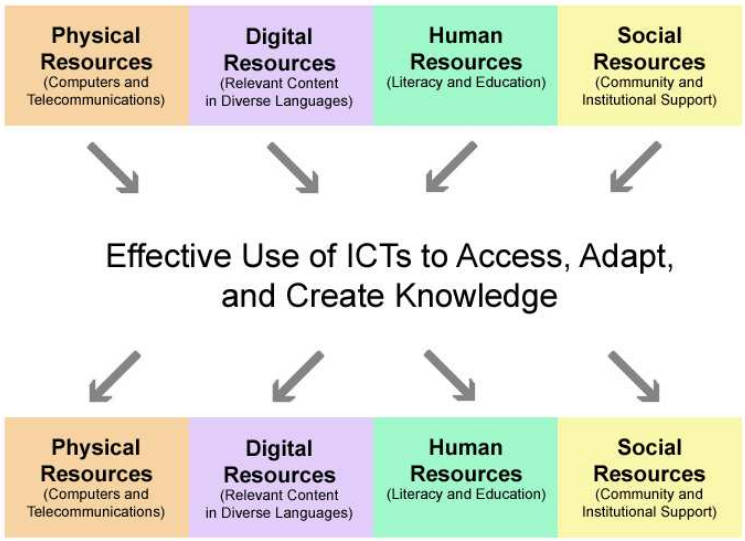

Figure 2.3: Effective Use of ICTs by Warschauer

The organisation Bridges which is a leading *Non-Governmental Organisation* (NGO) in the application of ICTs to economic and social development, links effective use to the term e-readiness[4].

> With the specter of the growing digital divide looming large, world leaders in government, business, and civil society organizations are harnessing the power of information and communications technology (ICT) for development. They seek to improve their countries' and communities' e-readiness – the ability for a region to benefit from information and communications technology. It is increasingly clear that for a country to put ICT to effective use, it must be 'e-ready' in terms of infrastructure, the accessibility of ICT to the population at large, and the effect of the legal and regulatory framework on ICT use. If the digital divide is going to be narrowed, all of these issues must be addressed in a coherent, achievable strategy that is tailored to meet the local needs of particular countries.

[4]See: http://www.bridges.org/e_readiness_assessment

2.3.2 Technology transfer/translation

In order to bridge the digital divide technology has to be "transferred" one way or the other. The developed countries are driving technological innovation while the developing countries are falling behind. The process of transferring the technological artifacts and "know how" from the developed to the developing world are frequently labeled *technology transfer*.

Different perspectives have been used to understand the technology transfer process. Nhampossa (2006) discusses three perspectives; diffusion, transfer channels and transfer life-cycle. The diffusion perspective argues that the adoption of technology tend to follow a S-shaped curve. This perspective give a prominent position to the individual adopter, and do not take the social system where the diffusion takes place into account. The transfer channels perspective describes technology transfer as being facilitated through channels like sale of technical artifacts, foreign investments and education. This explains technology transfer as a one way sequential process, and it suggest that the success or failure of ICT project can be explained by the effectiveness of the different channels. For the third perspective, the technology life-cycle perspective, I will give a more detailed explanation.

In their paper on donor-funded ICT transfer Baark and Heeks (1998) have derived a conceptual framework that they label *information technology transfer life-cycle*. This framework is visualised in Figure 2.4. Because of the regular infusion of new technology the process is depicted as cyclical and seen as a continuous process. Each cycle have up to five phases:

1. The technological requirements are identified and the different alternatives for new technology are surveyed. Based on these assessments the appropriate technology are chosen.
2. The chosen technology are purchased and installed. This often include some training and consultancy to assist in the installation.
3. Assimilation and use. The goal of this phase is to ensure that the people who work with the technology understand how the technology works, how to use it and how to maintain it.
4. Depending on the technology and the planned use of it there might be a need for adapting the technology. For context sensitive technology, which ISs most often are, this is a fundamental part of the project. For ICT specific projects this phase helps to build local ICT capabilities.

5. When the recipients master the technology they can transfer the technology to other organisations and make innovative changes to the technology, or even make new technologies.

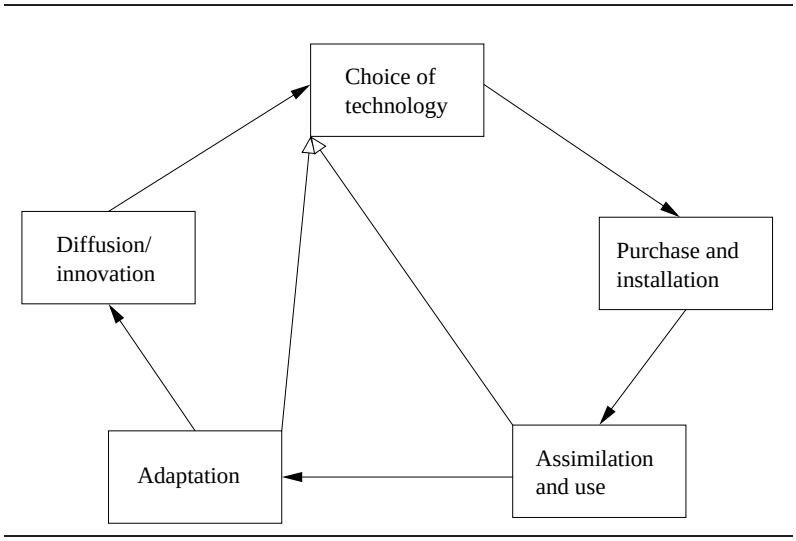

Figure 2.4: The Information Technology Transfer Life-cycle by Baark & Heeks

Nhampossa (2006) advocate a fourth perspective, the *technology translation* perspective. He argues that the previously mentioned three perspectives are limited due to the following reasons:

- Technology are treated as "black box". Technology is seen at a superficial level, with to little emphasis on the technical specifics.
- Technology transfer is seen as a one way sequential process. Technology is created in developed countries and transferred to developing countries to help them "catch up". This view tend to disregard the political brokering and negotiation needed to make things happen.
- The intra-organisational conditions are not taken sufficiently into consideration.
- The diffusion perspective sees individuals as de-linked from the socio-technical context and fails to recognise the technological learning which may result from the transfer process.
- The technology transfer process is seen as one giant step from developed to developing countries rather than series of incremental and

> interconnected steps. Technology developed in the west are often inscribed with western assumptions of rationality. This technologies cannot be replicated, but have to be translated.

Technology created in developed countries are designed for social systems quite different from the realities in developing countries. For strongly context sensitive technologies like health information systems this poses real problems. Heeks (2002) calls this design mismatch a *design-reality* gap, and use this to explain why most ISs in developing countries fail either totally or partially.

The technology translation perspective is seen as the process of cultivating sustainable networks. For a technology translation process to be successful the technology and the surrounding network must have the capacity to endure over time and space, in other words be sustainable.

> Technologies or systems become sustainable if they are institutionalized in the sense of being integrated into the everyday routine of the user organization. However, sustainable technology or systems need not only be institutionalized, but also need to be flexible in order to allow for changes as the user needs them.

(Nhampossa 2006, p.57)

Nhampossa defines technology transfer as a process defined by three point.

1. The initiative should be designed as an incremental and context sensitive process, carried out in rather small steps.
2. Translation represents an iterative and evolving long term process, having implications for both sustenance and scale issues.
3. Technology translation includes building and supporting heterogeneous socio-technical networks and ensuring indigenous capacity building.

A key characteristic of this definition, Nhampossa argues, is the need to balance flexibility and stability. Sustainable systems must be institutionalised and at the same time remain flexible enough to accommodate changes occurring over time and space. Relating to ST this need for balancing stability and flexibility can be expressed as stabilising social systems, but at the same time be able to adapt to changes in the social practices. ICTs are best suited

for stabilised social systems, but the social systems in developing countries tend to be more unstable. This lead me into thinking that the process of technology translation involves making the social system more stable and at the same time making the ICT more flexible.

Nhampossa (2006) further identifies four key influences that are influencing and influenced by the process of technology translation.

Legacy system and installed base Organisations frequently have legacy systems which are based on outdated technologies. This systems often have high business value because they have important business rules inscribed into them, and is an important part of the current social practices within the organisation. This systems can be expensive to maintain and have high inertia, so it is important to identify which legacy system that need to be changed and how to build upon and incorporate the legacy system into a strategy for change. In other words there is a need to cultivate the installed base.

Adapting the software to the local context Depending on how context sensitive the ICT is there might be a need for making changes to the software, both configuration changes and changes to the code. Therefore it is important to identify the context free and context dependent technologies, and which part of a program that is context free and context dependent.

Participation The importance of participation I have already described in section 2.2.

Balance between localisation and internationalisation Here internationalisation refers to the process of isolating the culturally specific elements of the software. Localisation refers to the process of infusing cultural or business specific elements into an already internationalised program. The balance between stability and flexibility need to be cultivated. To cultivate this balance generic functionality and changeable functionality must be identified.

The theoretical framework for technology translation is summed up in Figure 2.5.

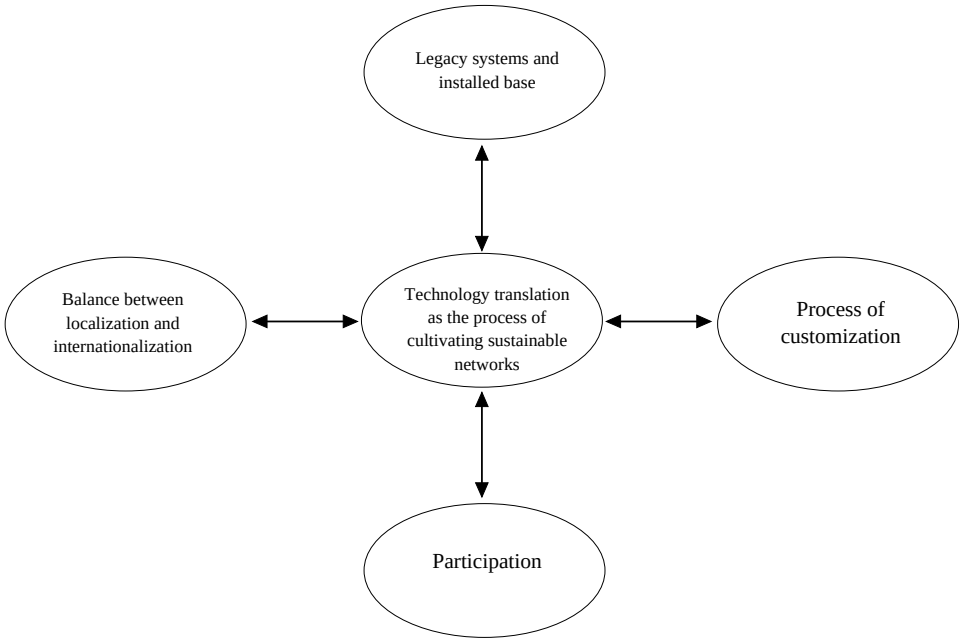

Figure 2.5: Factors influencing the technology translation process by Nhampossa

Part II

Method

Chapter 3

Methods

Now that I have framed my research within a predominantly social-science perspective I will move on to what kind of methods that I have been using in my research. My research takes place within a broader research network, so the research methodology is largely established. The methods I use is similar to what others in this research and development network have used previously. There are some differences, however, because I am not primarily researching the introduction of a health information system in a country or region. I am primarily researching the use of FLOSS in the context of HISP and Ethiopia. The cornerstone methodology of the HISP network is *action research*, this will also be my cornerstone methodology. This methodology will be applied within case studies.

In this chapter I will first explain the methodology I will rely on and then I will move on to how I will apply this methodology in my research setting. At the end of the chapter I will identify some limitations with my research approach.

3.1 Research methodology

As mentioned in the introduction to this chapter, my cornerstone methodology will be action research, so I will give a quite lengthy description of this methodology, and relate it to IS and FLOSS. Then I will give a short definition of what a case study is, because case studies are my primary source of firsthand information.

3.1.1 Action research

Modern *Action Research* (AR) has its heritage from social psychology related research conducted in the 1940s. AR have since its inception developed into a multifaceted research methodology. I am going to use the definition made by Greenwood and Levin (1998) as a basis:

> AR is social research carried out by a team encompassing a professional action researcher and members of an organisation or community seeking to improve their situation. AR promotes broad participation in the research process and supports action leading to a more just or satisfying situation for the stakeholders.

Unlike conventional social science research that try to avoid any intervention in the research setting, AR demands intervention. AR do not primarily seeks to create general theories and to prove them true or false, AR shifts the focus away from general theories in favor of local relevance. AR is grounded in practical action aimed at solving real life problems while carefully informing theory. To inform theory AR involves a process of deliberate and systematic reflection, and generally require some sort of evidence to be presented to support conclusions. AR is inquiry that is done *by* or *with* the members of an organisation or community, never is research done *to* or *on* them (Herr and Anderson 2005).

AR is, as previously mentioned, a multifaceted methodology which is practised in a number of different ways. According to Greenwood and Levin (1998) there are, however, three basic commitments that must be present for a process to be called AR; *research*, *participation* and *action*. I will use the reminder of this section to investigate this three commitments.

A powerful motivating commitment in AR is to change an organisation or community into a more empowered, democratic and liberated state. The focus on change is characteristic of AR, actions are made to change the organisation and community in a way that the stakeholders have agreed on. AR focuses on action to create change in a social system, this is highly political so an AR practitioner have to deal with politics. Herr and Anderson (2005) frames the political considerations in AR within, among others, micro- and macro-politics. Micro-politics is the politicising that takes place in the local context of an organisation or community.

> Micro-politics includes the behind-the-scenes negotiations over material resources, vested interests, and ideological commitments. More often, micro-political struggles are over such things

> as professional jealousy, power differences in the organizational hierarchy, the allocation of space and other resources, gender and racial politics, and so forth. ... [Micro-politics] is as much about what doesn't get said as it is about what does.

(Herr and Anderson 2005, p.65)

Problems in the local context are often related to broader social forces and problems. It is therefor important to identify macro-political factors influencing the context where the AR is conducted. The motivation to initiate AR research often comes from broader social and political issues. HISP was for instance initiated as part of a broader political agenda aimed at promoting equity after the end of the apartheid regime in South Africa.

In AR the stakeholders in a research setting and the researcher will, usually, all be participating. How much stakeholders will participate will vary. In some AR research the stakeholders will be co-researchers, they will participate in the whole research cycle from identifying problems to reflectively identify learning. Many forms of AR, *Participatory Action Research* (PAR) among them, strongly emphasises this. It is important that the researcher and the stakeholders participate on level turf and develop a relationship based on mutual trust. By involving the stakeholders in identifying learning from an AR study, the creation of knowledge is made more democratic, and it is more likely that this learning would be used to inform new actions by stakeholders.

As mentioned in connection with PD in section 2.2 effective participation can be a challenge to instigate, this challenges are relevant for participation in AR too. In many settings it is not feasible to form a close working team of researchers and the stakeholders. It might only be feasible to involve stakeholders in some of the phases of the research, like in action planning and action taking, but not in identifying learning. Neither is it necessary to form a formal team, participation by stakeholders can be achieved in less formal ways. Some level of participation do have to be present, however, even if it is limited to the researcher participating in the problem domain, be it an organisation or community.

The relationship the researcher have with the organisation or community under study is something Herr and Anderson (2005) brings attention to. They use a scale they call *the continuum of positionality*. This scale goes from being an insider to the research setting to being an outsider. The degree of participation is highest at the middle of the scale. On the far left of the scale the researcher is an insider to the research setting, and the focus is on the research's own practice in relation with the research setting. Two other

form of insider studies are when an insider collaborates with other insider and when insider(s) collaborates with outsider(s). In insider studies the research is initiated by insiders.

At the middle of the continuum of positionality, where the degree of participation is highest, are the insider-outsider teams where the outsiders and insiders work together on level turf and have mutual trust. This is close to an ideal form of PAR, but it can take years to develop this kind of relationship, so it is generally not feasible within the constraint of a master thesis. More feasible is it for outsiders to collaborate with insiders, but where the researcher is clearly seen as an outsider. On the far right of the scale are the research where outsider(s) studies insiders. This is the form of AR that is closest to conventional social science research, but the researcher engage more closely with the study's participants.

The insider-outsider positionality is one of many ways to think about positionality.

> Positionality occurs not only in terms of inside/outside, but also in terms of one's position in the organizational or social hierarchy, and one's position of power vis-à-vis other stakeholders inside and outside the setting.

(Herr and Anderson 2005, p.41)

AR is partly based on the belief that action and participation brings understanding. By introducing changes into a social process and observe the effect, knowledge about the local context can be deduced. AR do not rely on any specific methods for capturing data during the research. The researcher, perhaps in collaboration with the stakeholders, can choose whichever methods, qualitative or quantitative, found to be appropriate in the research setting. No matter which methods that eventually get chosen it is necessary to pay close attention to the quality of the data gathered, and be able to assure the correctness of interpretations made from the data. The researcher have to be able to demonstrate that the interpretations are more likely than alternative interpretations.

In complex and changing social settings the degree of control needed to get reliable data by conventional research is often unattainable. Even if the problem domain is broken down into sub-problems that can be investigated under controlled condition, the researcher risk that the sub-problem will become irrelevant in the changing social setting. AR is often portrayed as a cyclical process like the one in Figure 3.2. AR attain rigor and flexibility by going through research cycles. The methods and research question can be refined in response to realities in the research setting.

> Most conventional research methods gain their rigour by control, standardisation, objectivity, and the use of numerical and statistical procedures. This sacrifices flexibility during a given experiment – if you change the procedure in mid-stream you don't know what you are doing to the odds that your results occurred by chance.
>
> In action research, standardisation defeats the purpose. The virtue of action research is its responsiveness. It is what allows you to turn unpromising beginnings into effective endings. It is what allows you to improve both action and research outcomes through a process of iteration. As in many numerical procedures, repeated cycles allow you to converge on an appropriate conclusion.

(Dick 1993)

A common notion in the social sciences to attain quality in data and interpretations is through the notion of *triangulation*. Simply speaking this is about looking at a problem from multiple perspectives. By using multiple information sources and multiple methods the quality of data will most likely be better. It is also necessary for the researcher to address his or hers own biases and how this can affect the research. The researcher have to be self-reflexive.

Goals of Action Research	Quality/Validity Criteria
1) The generation of new knowledge	Dialogic and process validity
2) The achievement of action-oriented outcomes	Outcome validity
3) The education of both researcher and participant	Catalytic validity
4) Results that are relevant to the local setting	Democratic validity
5) A sound and appropriate research methodology	Process validity

Table 3.1: Anderson and Herr's Goals of Action Research and Validity Criteria

In Table 3.1 Herr and Anderson (2005) have proposed five validity criteria for AR and linked them to the goals of AR. What constitute evidence and validity in AR are still disputed, but they offer this criteria as tentative criteria. I will shortly describe this criteria here.

Process validity refers to how the research is conducted. Important here is the cyclic and ongoing problematisation of the practices under study, the appropriateness of the methods used and the quality of the relationships built during the study. To have a good basis for the assertions made during the study, the notion of triangulation is important.

Dialogic validity. The research data, methods and interpretations should be subject to peer review. The reviewer of the research can be stakeholders in the research setting, colleagues or friends familiar with the research setting, or fellow action researchers. It is important that the reviewer are critical in reviewing the research.

Outcome validity is the extent to which action occur, which leads to a resolution of the problem that lead to the study. Action researchers have a double burden in that they have to focus on both action and research. The research must be of high quality as the same time as the action researcher have to move participants towards a successful action outcome.

Catalytic validity is the degree in which the researchers and stakeholders have moved towards a better understanding of the research setting, understanding which have the potential to transform the reality of the research setting. As with PD the concept of mutual learning is important, both the researchers and the stakeholders learn from each other and from reflecting on the problem domain.

Democratic validity is the extent to which research is done in collaboration with all stakeholders in a research situation. If the study is not done collaboratively the researcher must take multiple perspectives and interests into account. Is AR used to improve the situation of some stakeholders at the expense of other stakeholders? The degree to which the research is relevant to the local context is also important to democratic validity.

AR is rooted in the local context and focus on what is relevant locally. The question is then how lessons learned in a local setting can be apply to other settings. This is the question on how generalisable research findings in AR are. General theories have to apply in every local setting, so AR could be used to test general theories. This is not the motivation to use AR, since AR focuses on change and flexibility and not on challenging general theories.

Greenwood and Levin (1998) change the question about whether research results from AR is generalisable to a question about transferability of AR research results. This places the responsibility of justification on the ones who seek to apply the research result. This looks somewhat similar to the notion of technology transfer in section 2.3.2. Perhaps we should talk about knowledge translation as well as technology translation?

> We argue that AR-developed knowledge can be valuable in other contexts other than those where it is developed, but we reject

> the notion that the transferability of knowledge from one location to another is achieved by abstract generalizations about that knowledge. Transferring knowledge from one context to another relies on understanding the contextual factors in the situation where the inquiry took place, judging the new context where the knowledge is supposed to be applied, and making a critical assessment of whether the two contexts have sufficient process in common to make it worthwhile to link them.

(Greenwood and Levin 1998, p.79)

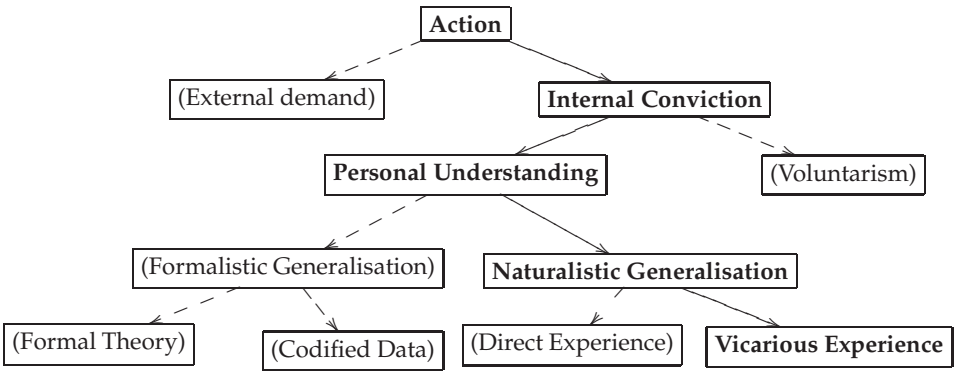

Figure 3.1: Robert Stake's Evolutionary View of Change

Herr and Anderson (2005) finds Robert Stake's notion of *naturalistic generalisation* a compelling way to think about how AR is taken up by other practitioners and researchers. Figure 3.1 is a model of Stake's view of change. Change is based on action and action is taken either because of external demand or internal convictions. Coercive external demand is often successfully resisted by practitioners, most lasting change takes place through internal conviction. Conviction is again based on personal understanding and voluntarism. Voluntarism are personal feelings, values and faith. Formalistic generalisation are the propositional knowledge common in academia. Naturalistic generalisation are generalisations made from experience. This experiences are either direct or vicarious, that is "second-hand" experiences. Narratives are one way to convey vicarious experience.

Action research in IS

Toward the end of the 1990s AR began to grow in popularity in IS research circles. AR is rooted in action and research, and IS is an applied field which should focus on research relevant for ICT practitioners and users. This makes AR a good fit for research in IS.

Baskerville (1999) list some specific forms of AR that they deem to be valid for research in IS. This list is given in Table 3.2 along with a reference to relevant literature explaining that form of AR. The references are given by Rose (2000). In addition to this list it should be mentioned that the Scandinavian approaches described in section 2.2 follow an AR approach. In the Scandinavian projects mentioned theory building was done based on the projects conducted.

Forms of Action Research	**Reference**
Canonical Action Research	(Baskerville 1993)
IS Prototyping	(Kyng 1991)
Soft Systems Methodology	(Checkland and Holwell 1997)
Action Science	(Reponen 1992)
Participant Observation	(Jepsen et al 1989)
Action Learning	(Naur 1983)
Multiview	(Avison and Wood-Harper 1990)
ETHICS	(Mumford 1983)
Clinical Field Work	(Hammer and Champey 1993)
Process Consultation	(Coad and Yourdon 1991)

Table 3.2: Forms of IS Action Research

AR is indeed similar to organisational consulting and ICT consulting. Baskerville (1999) points out the need to distinguish between AR and consulting. AR is different from consulting in its commitment to research and the research community. A consultant is expected to get things done, and to suggest solutions based on experience. An AR researchers have to consciously collect data, analyse and reflect, and based on this the action researcher have to write papers founded within a theoretical framework. AR usually gives more focus to collaboration, while a consultant is usually hired to get a job done or to give an outsider‘s unbiased view of the organisation.

AR is as previously mentioned frequently portrayed as a cyclic process. The most prevalent model of this cyclic process is given in Figure 3.2 (Baskerville 1999). The *client-system infrastructure* in which the five phased cycle takes place, refers to the specifications and agreements that the AR

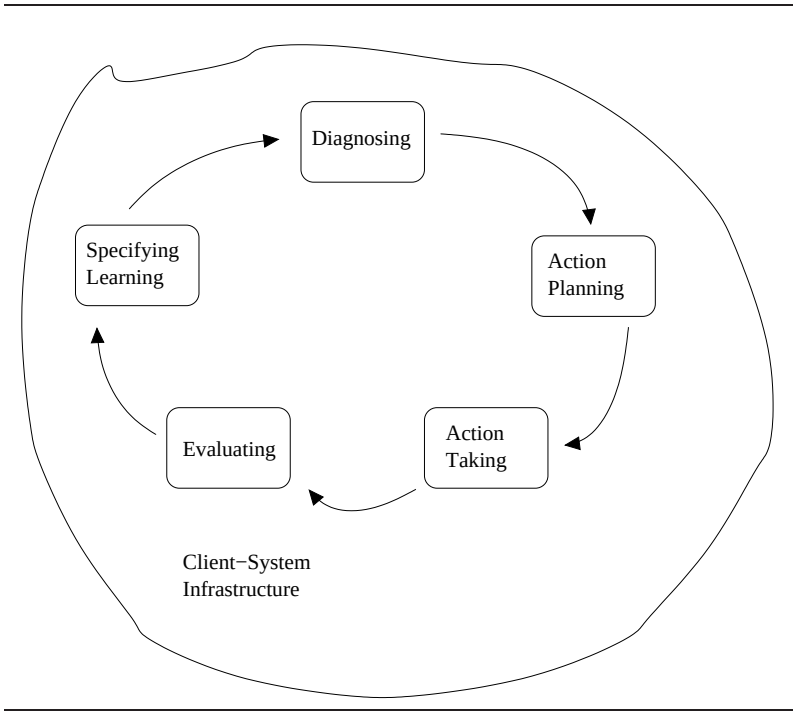

Figure 3.2: The Action Research Cycle

research is governed by. This can be an agreement with a client organisation or the rules governing participation and action in more open ended settings, like a FLOSS project.

Action research in the field of FLOSS

As a basis for his essay *The Cathedral and the Bazaar* (Raymond 2001) Eric Raymond conducted action research. He did not give it that name, but that is what he non the less did. He was handed over the fetchmail (at the time called popclient) project. Previously he had been exposed to Linux development and decided to test the lesson he had learned from Linux in the fetchmail project. Based on the experience he made from managing this project and from cooperating with the contributors, he wrote his essay.

I deem FLOSS projects to be fertile ground for conducting action research. Many, but not all, FLOSS project are very open to contributions. An action researcher can take on many roles in an FLOSS project. If the action researcher is a member of the project's core he has the position of an insider, possibly collaborating with people more or less affiliated with the project.

The action researcher can also be an outsider seeking to contribute to the project. Eric Raymond had the position of an insider cooperating with more or less outsiders. The core of the project was basically only Raymond himself.

Doing AR in FLOSS is similar in many ways to doing AR in IS. AR in IS frequently happens within the confines of one or more organisations, while in FLOSS the participation and cooperation between stakeholders and researchers will most likely be more virtual. The client-system infrastructure will then consist more of rules for contributing and channels for cooperation, than it will consist of contracts and agreements.

3.1.2 Case Study

A case study, which is an in-dept exploration of one situation, is a common qualitative method used in the social sciences.

> ...the distinctive need for case studies arises out of the desire to understand complex social phenomena. In brief, the case study method allows investigators to retain holistic and meaningful characteristics of real-life events – such as individual life cycles, organizational and managerial processes, neighborhood change, international relations, and the maturing of industries.

(Robert K. Yin 2003)

Case studies are often exploratory in nature and is a good fit for studies of little known problem domains. Case studies have been criticised for being a poor basis for generalisations, especially formalistic generalisations. Case studies are good for naturalistic generalisations, however. By basing assumptions on a number of case studies and by doing longitudinal case studies, stronger and stronger evidence for assumptions can be given. When the problem domain is better understood other kind of research methods like surveys and experiments can be useful.

3.2 My research approach

When I started to work on this thesis I did not have any clear understanding of neither FLOSS nor HISP. I fact, I had only limited knowledge about

action research and IS theory. The beauty of action research is that the cyclical nature allows you to start out with a "fuzzy" research question and a limited understanding of the problem domain. During the advancement of my research I could constantly refine my questions and methods, and thereby give better answers.

My motivation for this thesis was to increase my understanding of FLOSS and developing countries, and how FLOSS can benefit developing countries. To do this I have read extensively about FLOSS, HISP, developing countries and Ethiopia. Most importantly, however, I have participated in the development of a prototype HIS in Tigray and the development of the next major version of DHIS.

First I will give a short description of the two cases I have participated in and state my position and biases relating to this cases. Then I will give an outline of the concrete methods I have been using during my research. I have framed my research within structuration theory so I will give an description of how I will use this as a framework for my research. Last I have to specify what I understands to be the limitations in how I have conducted my research.

3.2.1 Working in the Tigray HISP team

This case study was conducted from July until the end of October 2004. The case study took place in Ethiopia. In Ethiopia I worked together with a team consisting of me and three others. The three others where Ethiopian nationals, as a Norwegian national I was the only foreigner on the team. Of the members in our team, one was a PhD student and the other three of us were master students. The research setting was in the primary health care system of Tigray. Ethiopia is a very ethnically diverse country, so all of us on the team was more or less foreign to the region of Tigray. Most likely I was more foreign than the others in our team, I do not speak Amharic and I am a white man from a developed country.

Our position in the relationship with the Tigray health bureau and the two districts we worked in was clearly an outsider team collaborating with insiders. The collaboration took shape through the forming of a team to decide on which data elements should be included in DHIS, and on which reports the system should be able to produce. During the training sessions we got constructive suggestions on how to improve the prototype we made. An important part of our purpose in Tigray was to develop an evolutionary prototype based on DHIS for the primary health system in Tigray. Our main purpose was to help health workers and management to

collect more reliable and useful data. A purpose the stakeholders naturally shared, at least in principle. We did not have time to develop a close relationship with the stakeholders, we spent in the vicinity of four months on this case, only two of them were spent in Tigray.

In the team I was an insider, but because I was the only one in the team not being an Ethiopian national I was in an other sense an outsider too. All on the team spoke adequate English, so language for internal communication was not a significant problem, except for some boring dinner conversations conducted in Amharic. I don't speak Amharic so all conversations conducted in Amharic are invariably boring. Language was more of a problem for my relations with the stakeholders, information will invariably get lost when a discussion is translated or retold after the discussion is finished.

Our case study in Tigray was only one of several AR case studies conducted in Ethiopia. Ethiopia is a node in the HISP network and there was activity in five regions when I was there, with different teams in each region. I will explain more about the HISP network in chapter 6.

3.2.2 Participating in the development of DHIS 2

This case study have been conducted part time from the start of 2005 until the start of 2006. I do not regard my participation in this project as finished yet, but because I had to start writing my master thesis I stopped to be active in this project from early 2006 and on. Hopefully I will be able to participate more when I am finished. This project is more in line with how FLOSS projects of some size are driven.

For reasons given in chapter 10 it was decided to do a total reimplementation of DHIS. The early planning for this huge undertaking had already started some time before I signed up to write my master thesis as part of the HISP network. As I am a computer scientist who likes to program I found it appealing to work on this reimplementation. After coming back from Ethiopia I therefore started to participate in this project. In the first semester of 2005 I participated in this project through a graduate course, after that time I still spent some time coding but did not actively participate in the virtual community of the project.

This case study is more of an insider or practitioner study. During the course of the case study I contributed somewhat to other projects, in this setting I was more of an outsider. However, the perspective in this case study is mostly that of a practitioner. My purpose for doing this case study was to learn more about FLOSS through action and participation. I have

to state that I was sympathetic to FLOSS from the outset of my research. To hand out code for free (as in freedom) use I do regard as a sympathetic thing to do. I had also previously been exposed to FLOSS software, like Linux, which I found fun, and sometime frustrating, to work with. This frustrations forced me to learn more about the inner working of a computer, which is a good thing for a computer scientist to learn.

3.2.3 Methods for data collection

In AR it is best to be flexible in the choice of methods used for data collection, question that demands an answer constantly appear when going through the research cycle. Some of this question are best addressed with the use of qualitative data, others can be better addressed with quantitative data. Most of the data collected are of the qualitative sort, but for the investigation of FLOSS in Ethiopia and for the investigation of the DHIS 2 project, I have collected some limited quantitative data. To collect quantitative data is a time consuming process, luckily I can use quantitative data others have collected by using existing statistics.

Observation, participation and training

The role of an action researcher is not that of an objective observer, but an action researcher still observes. Observations, and simply talking to stakeholders in the day to day work of training and making the prototype, or as I did in the DHIS 2 case, sending e-mails, is an important source of information.

In the Tigray case, the negotiations we had with the regional health bureau to set up a client-system infrastructure for our research was an important source of information. In the meetings we had with different authority persons in the bureau I took some scrappy notes as time and translation allowed. By combining this notes with my memory and the memory of other team members, I could write it up in my diary. The meetings could have been recorded, but the stakeholders did not approve of it.

We conducted training sessions at the bureau and in the two health districts. The training session at the bureau lasted over one week, in the district we had a two day training session at each site. I participated in the training session at the bureau and in one of the districts. Through this sessions I got valuable feedback. The decisions on how to conduct this training sessions was done in a partly participative manner, but the unit leader had a bigger say than the computer clerks.

In the DHIS 2 case, most of the communication were done virtually which means that the communication is more traceable. I have saved the e-mail I have sent to the different projects I have interacted with. E-mails sent to e-mail list are stored in e-mail archives available thought the public Internet. Some information are made available through the DHIS 2 Wiki. I also logged conversations I had through instant messaging.

Literature, electronic archives and statistics

My primary source of secondhand information, was information accessible through the Internet. For information relating to FLOSS this is a natural place to look. For books that were not electronically accessible I sometimes used the university library, at the University of Oslo.

For information relating to the Tigray case I used articles I found searching the Internet, international news media and statistics available from different organisation working with Ethiopia or developing countries in general. In Table 3.3 I have made a list of the most important sources of information.

Information Source	URL
Wikipedia	http://en.wikipedia.org/
UN	http://www.un.org/
UNDP	http://www.undp.org/
UNMEE	http://www.unmeeonline.org/
UNICEF	http://www.unicef.org/
The World Bank	http://www.worldbank.org/
CIA World Fact Book	https://www.cia.gov/cia/publications/factbook/
allAfrica	http://allafrica.com/
The Reporter	http://www.ethiopianreporter.com/
National Election Board of Ethiopia	http://www.electionsethiopia.org/
Ethiopian News Agency	http://www.ena.gov.et/
Ethiopian Telecom	http://www.telecom.net.et/
BBC	http://news.bbc.co.uk/africa
Guardian Unlimited	http://technology.guardian.co.uk/

Table 3.3: Internet Information Sources for the Tigray case

For secondhand information relating to FLOSS I have used sources too numerous to list here. Important references are in the regular reference section. It is common in FLOSS projects to have publicly available e-mail archives, discussion forums, Wikis, issue tracking systems and other technologies used to facilitate cooperation. Most of the information I have used I got from regular books and articles about FLOSS. If I had time to go through some more AR research cycles I could have investigated more archived information.

Narrative - Recording a diary

During my stay in Ethiopia I regularly kept a diary. In the chaotic AR research setting of Tigray this was my most important method to record my experiences and to remember names and conversations. To get the most reliable information this should be done at the end of every day, or sooner if you need to write up some scarp notes while it is still fresh in memory. If the day was rather eventless there is limited need to write an entry in the diary, but it is a good habit to have. I have to admit, however, that as our case study progressed and more demands was laid on me, my diary suffered. In the beginning I wrote frequent diary entries, but the frequency became less as the case study progressed.

Automated collection of quantitative data

On all the computers that makes up the Internet there is stored massive amount of data. A lot of this data are of the qualitative sort, but it is also possible to collect quantitative data for analysis. Unlike surveys and questionnaires this do not need to involve lengthy processes of interviewing to gather the data. The data is already "out there" in a form accessible to a computer program. By simply using a program this data can be collected. If there is no program available fit to collect the needed data, it is possible to make one which do. Two papers I have read (Lancasshire 2001) and (Mockus et al. 2002) did this. Both made Perl script to collect and process data. I have done this in my research on three occasions. On one occasion I made a python script to search for Ethiopian web servers on Google and on the second occasion I made a simple Perl script to process Linux credit files. On the third occasion I made a Perl script to gather information from the Subversion repository used in the DHIS 2 development.

To gather information about the topology of the Internet and information about the hosts connected to the Internet there exist network analysis tools.

I have used two such tools `traceroute` and `nmap`. `traceroute` is a program useful for discovering the topology of the Internet. I used this program to find the international Internet connection of Ethiopia. `nmap` is a network analysis tool. This tool should be used with caution, it can potentially be ethically questionable because it is a tool that can be used to gather information that can be used to crack into a private networks. This tools is useful for discovering hosts, and to get information about them. I used this tool to find web servers running within the Ethiopian IP range.

Why I didn't do formal interviews

When I first came to Ethiopia I had a plan to do some formal interviews, of the semi-structured or the structured kind. This is a common method used to collect qualitative data. I soon discovered that I did not have sufficient knowledge of FLOSS to ask useful questions. I was a foreigner in Ethiopia so I had no idea of who I could ask. Ideally I should have prepared more before I went to Ethiopia, but there was a need for people in Ethiopia quite soon after I had volunteered to go to Ethiopia. Part of the cyclic nature of AR is the constant refinement of research questions and methods based on experience, so if I could go to Ethiopia now I would be able to ask useful questions. I made one semi-structured interview with one of the managers of the DHIS 2 project, Knut Staring. This was done mostly to verify my interpretations of the DHIS 2 project.

3.2.4 How I will use ST

I have placed my research within the theoretical framework of ST (described in section 2.1.2). Here I will describe how I will use this framework on a practical level.

Social systems in Gidden's definition are social practices replicated through time and space. This leads me into thinking that history is important to understand why the social systems are the way they are today. For this reason I will give relative (to the length of this thesis) lengthy historical accounts for especially FLOSS, but also for Ethiopia. By observing the social systems over an extended period of time it is possible get a better understanding of the social systems, and a better understanding of how and why they change.

Even if ST focuses on the replication and change of social systems over time and space I think it can be useful to give a snapshot image of the current situation in a social system. The model I have developed to give this snapshot

view clearly resembles Gidden's analytical model of the duality of structure (see Figure 2.1). In my model I seek to identify important modalities in the three dimensions contributing to the structures; signification, domination and legitimisation. In Figure 2.1 the focus is on the modalities and the images I have used are only meant to illustrate the model. In actual use I will not necessarily use images to visualise the modalities, more likely I will use textual descriptions. I will only use this model in my analysis of the broad social system of FLOSS. I did not get a sufficiently deep understanding of the primary health system in Tigray to model it in this way.

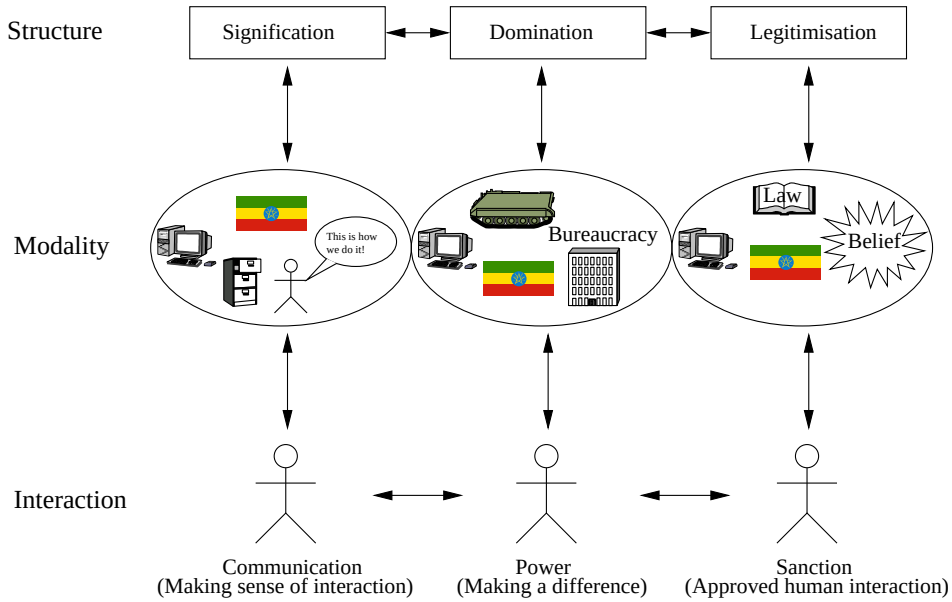

Figure 3.3: An analytical model loosely resembling the duality of structure

3.2.5 Limitations in my research approach

Doing formalistic generalisations based AR must be done with extreme care, AR do not naturally lend itself to formalistic generalisations. For naturalistic generalisations the responsibility for justifications is not mine, but the researcher using my research have to do the justification. In forming this thesis I have relied on other AR studies, which I hopefully have provided sufficient justification for. To make more formal generalisations the number of research cycles are important. In my study I have only went through one full circle in each case study. I have constantly refined my understanding of the research domain and I have made use of different

methods along the way, but this has only been within the phases of the circle in Figure 3.2. The reason that I have only went through one circle can be attributed to the time constrains of a master thesis.

I should also have subjugated my research and interpretations to more peer review. Unfortunately I do not have an extensive network of friends with knowledge about FLOSS, Ethiopia and developing countries. I have also been hard pressed for time so I have not prioritised searching for people willing to read through my thesis (which is not a small task). The evaluation and specifying learning phases have not been done in a participative manner.

Even if I had predominantly good relations with my team members and the staff at the research sites in Tigray, I was a foreigner. The people I worked with spoke English, but some of the meetings and a lot of the more informal conversations were conducted in Amharic. Because of language problems and because I didn't have deep knowledge about the culture of Ethiopia and Tigray, I might have missed some important clues.

To facilitate sustainability of our effort it is best with many research sites. This is more carefully explained in section 6.2. We only had access to two pilot sites in addition to the regional health bureau. The management of the bureau gave us access to only two districts. There were also several factors limiting the value of participation. There was a high rate of staff turnover, the bureau head was changed between our first and second visit. The staff at the bureau was also hard pressed for time in our first visit, an annual report on the health in Tigray had to be finished. Fractionating between the different departments in the bureau also put its limits on cooperation.

Because I didn't record the meetings we had with the bureau I might have missed some important information. I should also have been more rigorous in maintaining my diary and faster at writing up my scrap notes. Considering my limited experience and the chaotic conditions of the research setting I managed to keep a quite good record of my research.

My participation in the development of DHIS 2 too soon became a solo effort. I, together with an other master student, started on developing a plug-in framework for DHIS 2. The other master student participated while the graduate course we took lasted. This effort became too difficult for the state of the technology at this time, and because we were not seasoned Java programmer and had limited time. We had a too grand vision for our plug-in framework. After the end of the graduate course I was on my own, and I was swamped into the coding of the framework. This limited my participation in the project as a whole. My effort became isolated from the rest of the project. I should have been more chatty about what I was doing.

Part III

Background

Chapter 4

Short History of Open Source

The term *open source* was agreed upon at a meeting in 1998 by a number of key open source advocates. The values, development model, licenses and culture the term seeks to contain, has it's roots from several decades before 1998. The idea of distributing the source code of a program, with permission to change and redistribute it, is an idea that have it's root in the early history of computing.

In the technology oriented communities that cherished the idea of being able to change and improve software the term *hacker* arose. Unlike what many connects with this term, this people were not seeking to break into computer systems and making viruses. The term arose before there was anything called the Internet. The hacker culture was a driving force behind creating what to day is commonly called open source.

By presenting this narrative of the open source history I want to give some basic insight about the people, technologies and occurrences that have formed what is commonly called open source to day. While the practise of sharing code has been present from the birth of computing, the philosophies, theories and values forming the source sharing communities of to day, have appeared and been refined over the years. I will use the terms free software and open source interchangeably to mean the same thing, unless otherwise specified. My most important source of information for this chapter is (Weber 2004).

4.1 The early start of programming

The history of the electronic computer is short in the space of time. The first electronic Turing complete computers were made during and after world war two. Turing completeness is a term based on the universal Turing machine, which is a simplified model of a programmable computer. Of the earliest computers the German Z3 from 1941, the British EDSAC from 1949 and the American ENIAC from 1945 can be mentioned. The dawn of modern electronic programmable computing came in the time my parents were born.

In the early days of computing there were no meaningful distinction between what we to day call software and hardware, or between users and programmers. There was only the computer and the people that operated it. The first computers were operated through front panel switches, and permanent storage was often cardboard cards with holes representing 1 and no hole representing 0 (or possibly vise versa). The operator of the computer had to instruct the computer using machine language, this is now called a first generation language.

It is very cumbersome and error-prone to instruct the computer though machine language. Therefore the need for an abstraction was soon realised. The first level of abstraction was to give each computer instruction a textual name and using hexadecimal, octal or decimal numbers in place of digital numbers. The program that translate from the textual representation of instructions to machine language is called an assembler.

Machine language on a x86 processor

```
10110000 01100001
00000100 00001111
10100010 00001000
```

Intel assembler for x86 processor

```
SECTION .data
x: dd 0

SECTION .text
mov  al, 061h
add al, 00Fh
mov [x], al
```

Different computers have different instruction sets and therefore programs written in machine language and assembler were tied tightly to the hardware. The distinction between hardware and software became more clear as 3rd generation languages started to appearer in the 1950th. FORTRAN, LISP and COBOL are example of languages first developed in the fifties.

FORTRAN code

```
      Ix=97+15
```

The fifties were dominated by large mainframe computers. This computers were based on vacuum tube technology, and were large as an office and expensive as a small office building. One example is the IBM 705 which was a commercial computer launched in 1954. This computer was sold at an average price of $1.6 million.

In those days there were little meaning in selling programs. You had a program on, punch card, paper strips or magnetic tape that you feed into the computer. The computer industry had it's income from hardware. There was, however, a need for programs to run on this computers, because programs were seen as add-ons to the hardware it made sense for computer manufacturers to collaborate. Therefore a collaborative organisation called SHARE was started in the US This organisation practised source code sharing. This was not unusual, from the very start of programming loose associations of programmers from different companies shared code.

In the sixties cheaper computers based on transistors came on the marked, most notably by Digital Equipment Corporation (DEC). Connected to DEC was a user group called *DEC User Group* (DECUS) founded in 1961. DECUS, like SHARE, practised source sharing which was common at the time. Engineers from different companies would meet to exchange experience and source code.

The early history of open source is closely connected to what Eric Raymond call *the hacker culture* (Raymond 2001), which we will see in the following section. This was the group that continued source sharing after software became commercially interesting.

4.2 The three strains of hackerdom

Eric Raymond dates the start of the hacker culture as we know it to day to 1961, the year *Massachusetts Institute of Technology* (MIT) acquires it's first PDP-1. The PDP-X strain of computers were made by DEC and sold at considerably lower price than the IBM mainframes. The lower price on computer hardware allowed university departments and corporate research units to buy computers.

Computer power was still expensive and giving each programmer his own computer was not an option. There was a need to divide computer power

between users. One early solution to this was to execute programs in batch. A programmer would give a paper strip to a computer administrator who in turn assembled different paper strips into a magnetic tape. This tape would be given to the Central Processing Unit (CPU), and programs would execute in sequence. This did not utilised the processor time very efficiently. The processor would be idle while the program was read into memory from the tape, or when the result was written to tape or printer.

The solution to utilise the processor better was to overlap different jobs in time. While one job was waiting for the tape, an other job could be executed. If you could change between different jobs for one user there was no reason that you could not do this for many users. From multitasking in batch processing the idea of Time Sharing-System came. Time Sharing-Systems are important because it allowed each programmer to have access to his own terminal. To send source code though a compiler in a batch processing system, waiting for an hour only to find that there was a compilation error must be frustrating.

The access to a terminal gave the programmer freedom to do more experimenting and playing with the computer. This spirit of playfulness and experimenting with the possibilities offered by the computer is a basic value among hackers. According to Eric Raymond the term "hacker" originates from the computer culture at MIT. Other places where the hacker culture flourished in the early years were *Standford University's Artificial Intelligence Laboratory* (SAIL) and *Carnegie-Mellon University* (CMU).

This earliest strain of the hacker culture grew on time-sharing systems, like *Incompatible Time-sharing System* (ITS) and TOPS-10 combined with MACRO-10, and DEC hardware, most importantly PDP-10 machines. ITS was a time-sharing system developed at MIT, TOPS-10 and MACRO-10 were the *Operating System* (OS) and assembler made by DEC. The predecessor for the Internet, ARPAnet was primary a network of PDP-10 machines. Many prominent free software advocates, like Richard Stallman founder of Free Software Foundation, have their background from this strain of the hacker culture.

The hacker culture also found inroads into a business research laboratory called PARC in the XEROX company. The modern day LAN based on Ethernet have its origin here. The *Graphical User Interface* (GUI), with windows, menues and icons, was also invented here.

A second strain of the hacker culture that would later replace the PDP-10 strain of the culture started to emerge from 1969 and onwards. In 1969 Ken Thompson made the first version of the class of operating systems called Unix. A colleague named Dennis Ritchie invented the programming

language called C. Both were working for *Bell Telephone Laboratories* (BTL).

Unix was reimplemented in C during the seventies, which made Unix highly portable. Earlier system like ITS was made in assembly, which made ITS obsolete when the PDP-10 line of computers were discontinued. Because of juridical reason AT&T who owned BTL sold source licenses at nominal fee. The Unix strain of the hacker culture was spread out to universities across the US and Europe.

The third strain of the hacker culture was a part of the emerging computer revolution which was going to give computing to the masses. The first personal computers (PC) were marketed in 1975. Apple were founded in 1977. At the same time Commodore Corporation entered the computer industry. Commodore later acquired and developed the Amiga strain of computers. Commodore sold cheap machines and sold their computers mostly i Europe. The Commodore 64 and Amiga introduced me to the computer, though I mostly used it to play games, so I have a slight emotional tie to this machines.

When you turned on a Commodore 64 you went straight into a BASIC interpreter. BASIC became the language of the first computer hobbyists. While the Unix and PDP-10 strains of the hacker culture was based in universities, colleges and research institutions, the PC strain was the one that made it into the sleeping room of high school boys. The only early connection with FLOSS I can see from this strain is the practise of sharing BASIC source, and computer magazines often listed BASIC code which made your computer do interesting things. Shareware, where you give out the binary of a program and ask people to pay, became more common. Shareware is also referred to as nagware or donationware because they often display a message each time you start the program asking you to pay for the software. Shareware, however, is not open source, as no source is distributed.

Even if you could not get the source for a program the practise of sharing, or pirating, binaries was common among hobbyists. The hobbyists were more interested in experimenting and having fun with their computers than being efficient. The hobbyists naturally shared code and binaries because they were interested in making their computer do new and interesting things without reinventing the wheel. It was against this crowd that Bill Gates, one of the founders of Microsoft, sent his “open letter to hobbyists” accusing them of stealing software.

4.3 Multics, Unix and AT&T

I treat the development of Unix in a separate section because of two reasons. The Unix type of operation systems is the heritage to the most important FLOSS operating systems, the different BSD variants of Unix and Linux. Second the practice of source sharing have been common in the Unix community thought it's history. There are important lessons to learn from this history.

In 1964 researchers at MIT along with colleagues at *General Electrics* (GE) and Bell Labs, began a joint project to build a second generation time-sharing system. It was designed for high availability, seeking to provide a computing utility on continuous basis, like the telephone or electricity systems. In this way you could plug a dump terminal into a socket in the wall and you would be connected to a Multics driven mainframe.

The Multics project was too ambitious for the state of technology at the time. It didn't help that the tree collaborators had different goals for the system and that there was an awkward structure of decision making. The project decided to write Multics in a not yet implemented language, PL/I. PL/I proved difficult to implement. The project didn't deliver on time and was under-performing when it did. Multics never became a commercial success, but it contained many new ideas in computer science. Among this ideas were dynamic linking and a single level of storage (discarding the clear distinction between process memory and files).

Bell Labs withdrew form the project in 1969. Several researchers at Bell Labs felt, however, that they had learned important lessons from the Multics project. Lessons that could be used to build a new and simpler operating system. In the summer of 1969 Ken Thompson, a researcher at Bell Labs, stayed at home while his wife took their new born baby to California to visit the grandparents. Thompson had access to a quite old and not very powerful computer, a PDP-7. Using the four quiet weeks he had available he allocated one week each to writing a kernel, a shell, an editor and an assembler. The system Thompson had made he called UNICS, a intentional play on Multics. Unics was later renamed Unix.

To build an operating system alone in just one month Thompson had to put behind big system mentality, and build small neat tings. The doctrine of smallness and simplicity is at the core of the Unix philosophy. Later the important innovation of the pipe would help in modularising Unix. A pipe is a mechanism for *inter process communication* (IPC) allowing, among other things, the output of one program to be given to the input of an other program. This made it easier to make small specialised programs. A program

producing an unordered list of names could be piped into a program for soring thereby producing an ordered list of name.

```
Command to produce an ordered list of host names in the ifi.uio.no domain
host -l ifi.uio.no | sort
```

There was increasing pressure to document Unix as it turned out that Unix would become more than an experimental toy. Dennis Richie, an other important Unix developer at Bell Labs, and Thompson turned the necessity to document into a virtue, because clean and well designed programs are easy to document, while documenting an ugly piece of code makes it clear just how ugly it is. The first edition of the programmers manual established the Unix tradition of listing each subprogram with an "owner". The "owner" was responsible for writing and maintaining a subprogram.

Until 1973 Unix had only been deployed inside AT&T. This situation changed drastically after Thompson and Richie presented a paper on Unix to several hundred developers gathered at the ACM Symposium on Operating Systems in October 1973. After the paper was published in 1974 a flood of requests for copies flooded in at Bell Labs.

For a few hundred dollars license fee AT&T supplied the source code of Unix. The interest in Unix might have seemed like a business opportunity to AT&T but for a consent decree made in 1956. In 1956 AT&T entered a consent decree on an antitrust allegation stating that AT&T could not engage in manufacturing or sale outside of telephone, telegraph and "common carrier communications" services. The AT&T lawyers interpreted this to essentially mean "no business other than phones and telegrams". The AT&T wanted a clean statement that they were not seeking software as a business. Because a decree provision required Bell Labs to license patients at nominal fee, Unix was licensed at nominal fee. The early Unix licenses were minimal. The software came "as is" without royalties to AT&T, but also without support and bug fixes. This encouraged sharing of support and bug fixes among Unix users.

In 1973 the important decision of rewriting Unix in the new high level programming language C was made. The initial version of C was developed by Dennis Richie between 1969 and 1973. This was important because this made Unix much more portable. Most earlier operating systems were made in assembler, and had to be rewritten for every computer architecture. The portability of Unix have made it possible to port it to many different computer architectures.

Unix made inroads into many university computer science departments. Because the source code was available Unix was treated as a research and

learning tool. This created a good environment for experimentation in software development, creating many useful applications which were freely shared. In 1976 *Unix to Unix Copy Program* (UUCP) was made. UUCP became a "poor man's" networking for Unix. You could exchange files point-to-point over ordinary telephone lines.

UC Berkley became an important node in the Unix community in the seventies. From the time of the first Unix installation done at Berkley there was a tone of cooperation between Bell Labs and Berkley. Thompson took a year sabbatical from Bell Labs to work at Berkley from the autumn of 1975. A number of popular Unix tools were developed at Berkley, most notably a pascal compiler and the ex line editor. In 1977 this tools were put together by Bill Joy into a package called the *Berkley Software Distribution* (BSD).

The first releases of BSD up until 2.9BSD for the PDP-11 architecture and 3BSD for the VAX architecture, were not complete operating systems, but rater a package of Unix tools. After that time the BSD releases included the kernel, C library and utilities making it a full operating system. A lot of code in the kernel, C library and utilities included source which required a AT&T license to use. This was no problem because you could get this license at nominal fee. This changed in 1983 when AT&T was forced to break up the company because of legal reasons. Now AT&T was not restricted from entering the software industry and the price of the Unix source license exploded to cost in the $100,000 range in 1988. Before the mood shift in, and breakup of AT&T, AT&T had happily shared source code in exchange for contribution and bug fixes.

In 1989 Berkley released a package without any AT&T code containing the increasingly popular TCP/IP stack for BSD and a number of other tools. This package where called *Networking Release 1* (Net/1). Net/1 became immensely popular and inspired Keith Bostic to bring up the idea of reimplementing as much of the utilities contaminated with AT&T code as possible. This was a huge undertaking so Bostic consciously designed a public, voluntary, Internet based development effort to write the C library and the hundreds of utilities needed by BSD. Kirk McKusick and Michael Karels had agreed with Bostic that if he wrote the utilities, they would work on the kernel, partly believing that Bostic would never be able to do it. This proved wrong so they started to work on the kernel. This complicated job was more or less finished by spring 1991, only missing 6 files still containing AT&T code. This files were deemed to hard to rewrite, so this almost complete operating system was released under the name of *Networking Release 2* (Net/2).

William (Bill) Jolitz ported Net/2 to the Intel x86 architecture and wrote the

six remaining files betting that x86 would evolve quicker than competing architectures. This release, dubbed 386/BSD, was released with a license allowing free redistribution and modification as long as the credit file was kept intact (a BSD style license). People started to contribute to 386/BSD about the same time as the graduate student Linus Torvalds was trying to find a Unix-style operating system for his x86 PC. Having no *World Wide Web* (WWW) he did not find 386/BSD, so he started on making his own kernel which became Linux (see section 4.6). 386/BSD and 4.4BSD-light have evolved into FreeBSD, NetBSD and OpenBSD (see figure 4.1). The proprietary versions of Unix went along a different path, with multiple incompatible competing versions, which I will not cover here.

4.4 The rise of the Internet

ARPAnet is usually coined to be the predecessor of the Internet, and indeed most of the technological heritage are from this network. ARPAnet in its first incarnation, was developed during the sixties based on ideas about packet switched networks. In packed switched networks transmission is divided into discrete packages individually sent over shared transmission lines. The ARPAnet project was founded by *US Defence Advanced Research Project Agency* (DARPA). During the seventies the network was expanded to include American universities, defence contractors and military institutions. In 1975 there were 57 packet switching nodes connected to this network, with peripheral nodes in Hawaii, London and Norway. In 1981 this had increased to 213 nodes. Up to four hosts could be connected to a packet switching node.

During the seventies the ARPAnet was dominated by PDP-10 computers and used a protocol stack called *Network Control Program* (NCP). ARPAnet was a *Wide Area Network* (WAN) of its own and was not an internetwork, which to days Internet is. An internetwork is a network between networks. Unix had point-to-point networking though UUCP. With UUCP, Unix users could exchange mail point-to-point which lead to the creation of servers where you could send and receive messages like on a bulletin board. This where called Usenet. UUCP formed a network separate from ARPAnet. In addition there were a chaotic number of different network technologies in the US and Europe. Some examples where the X.25 based SERCnet(1974), and CERnet(1976) with its own protocols. X.25 is an ITU-T (The standards body of *International Telecommunication Union*) standard protocol suite. X.25 was designed to provide a packet-switched WAN on the analog telephone system. This protocol suite is obsolete, but X.25 based networks still remain some of the only available reliable links to the Internet in many portions of

the third world.

The most important reason that ARPAnet is seen as the predecessor of to days Internet is the TCP/IP protocol suite. TCP/IP was designed so that it could be used over many different kind of networks as long as the intermediate routers and end stations spoke IP. IP packets could therefore travel over may different kind of physical networks and rely on existing protocols for lower layer communication. ARPAnet switched over to TCP/IP in 1983. 4.2BSD with fully integrated TCP/IP networking where released in 1983, increasing both TCP/IP's and Unix's popularity. During the eighties and early nineties more and more X.25 based networks started to route IP traffic. *Internet Service Providers* (ISP) providing dial-up connection to IP networks replaced the networks of UUCP connected hosts. This connection of many different networks by means of IP have formed the Internet of to day.

The *Open Systems Interconnection* (OSI-stack) was an effort to standardise networking that was started in 1982 by the *International Organization for Standardization* (ISO). The OSI protocol stack was considered by many to be too complicated and was almost impossible to implement. The OSI protocol stack specified every layer in the protocol stack demanding existing protocols to be replaced. The OSI stack was eclipsed by the simpler and more pragmatic TCP/IP protocol stack. A lot of the functionality in the OSI protocol stack have later been implemented by other means using the TCP/IP stack.

4.5 Free Software Foundation

The MIT Artificial Intelligence Lab was in the 1960s and 1970s a major center for development of software. At this time when the hacker culture at the lab flourished, a undergraduate student named Richard Stallman started to work in 1970. He got a part time job at the lab, while at the same time he was studying mathematics at Harvard. Stallman did excellent achievements in his mathematics studies, and as a spare time activity he sought out computer labs asking for manuals and executing trial programs he made. This activity lead him to MIT.

The culture at MIT was stark contrast to the culture at Harvard. At Harvard access to computer terminals was given according to academic rank, and undergraduate students like Stallman often had to wait until midnight to get access to a terminal. At the same time terminals were sitting idle, locked inside professors offices. At the MIT AI lab they had a "first come,

first served" policy. This policy owed much to a tightly knit group calling themselves "hackers". The hackers spoke openly about changing the world through software, and viewed with disdain any obstacle that prevented them from fulfilling this noble goal. Chief among these obstacles were poor software, academic bureaucracy, and selfish behavior. This group of people even broke into professors offices to "liberate" locket in terminals (Williams 2002). This group of people, belonging to the ITS/PDP-10 strain of hackerdom, would have a profound impact on Stallman.

During the first decade Stallman was working at the AI lab the hacker culture had a strong standing. It was perfectly normal to share the source code of your work. As Stallman describes it:

> We did not call our software "free software" because that term did not exist yet, but that what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.

Nearing the 80s things started to change as software increasingly became commercially interesting. Companies wanting to profit on software started to require programmers to sign *None Disclosure Agreements* (NDA). This prevented the programmer from sharing the code with others. Stallman experienced this when the AI lab got a new printer from Xerox. Because the printer suffered from paper jams Stallman wanted to change the code driving the printer. He wanted to make the printer send a message to everybody in the printing queue that a paper jam had occurred, so that it could be fixed. When he asked a professor at Carnegie Mellon, who previously had been working on the code for the Xerox printer, if he could get the source code Stallman where refused. This infuriated Stallman.

During the first years of the 80s a lot of things would happen to the hacker culture at the AI lab, so much that Stallman felt his "home", the hacker culture he strongly identified with, was threatened. A lot of hackers at the lab were hired away to a start-up company named Symbolics. This company's software was based on free software, but they would not share their improvements of the software. DEC also discontinued it's PDP-10 line of computers. ITS where written in PDP-10 assembler, therefore ITS became obsolete. Because so many of the AI lab's hackers were hired away, porting ITS was not an option.

All the controversy around the AI lab hackers being hired away, lead Stallman to believe his commune was gone. Not being able to get the source

code for the Xerox printer Stallman had considered only a practical nuisance, but now he stared to believe that this was a moral issue. The right to share and modify a program was a question of freedom and a question of being a good neighbor. This he would later express in the following four freedoms:

1. The freedom to run the program, for any purpose (freedom 0).
2. The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbor (freedom 2).
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

It is worth pointing out that free do not mean "no price", this freedoms do not prevent anybody from selling copies.

In January 1984 he left his job at MIT to start the GNU project. GNU is a recursive acronym meaning GNU's Not Unix. He reasoned that in order to promote free software a free operating system had to be made. As he puts it (Stallman 1999):

> With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers – and invite anyone to join. And anyone would be able to use a computer without starting out by conspiring to deprive his or her friends.

An operating system consist of many parts, a kernel, system libraries, a shell and tools. The operating system also needed to be made in a higher level language to make it portable.

After searching for some time after a free C compiler without much success he started on GNU Emacs. Emacs was an editor originally developed by Stallman and many other hackers at the AI lab. Emacs had forked in many different version, not all where free. Stallman initially started to borrow code from GOSMACS, a Lisp based Emacs. Lisp is a language originally

developed at the AI Lab to assist in AI research, and became a favored language at the lab. The author of GOSMACS had sold the copyright to a company named UniPress. When UniPress threatened to enforce the copyright, Stallman decided to reverse engineer GOSMACS. In early 1985 GNU Emacs was beginning to become usable. Now the GNU project had some code to show, and request for GNU Emacs started to come inn. Stallman and a few colleagues started *Free Software Foundation* (FSF) to take care of the business side of the GNU Project shortly after the release of GNU Emacs. GNU Emacs is still very much alive to day, in fact I write my thesis using this editor.

The name Emacs is short for "editing macros". Emacs originates from TECO, TECO is short for "Text Editor and COrrector". TECO was cumbersome editor to use. You edited a document by typing series of editing instructions, making TECO a cross between an editor and a programming language. In the late seventies Stallman revised a feature in TECO called Control-R which switched TECO into a keystroke-by-keystroke mode (when you typed a `G` a `G` would be inserted into the buffer. Stallman revision made TECO execute a number of instructions stored in a file, a macro, when a two key combination was typed. The user could now make two key combinations execute some useful instructions of he's own choosing. This feature inspired an explosion of innovation. After a couple of years the number of macros had become a problem, sitting down at an other user's terminal it could take an hour to understand what the other user's macros did. Guy Steele took it upon himself to solve this problem. Stallman and Steele started an effort of standardisation. This collection of macros was named Emacs.

The Copyright Act made in the U.S. in 1976 extended copyright to software programs. Companies and individuals could now copyright the "expression" of a software program but not the "actual processes or methods embodied in the program." Stallman initially viewed this development with alarm, but proponents of copyright argued that (Williams 2002):

> Using copyright as a flexible form of license, an author could give away certain rights in exchange for certain forms of behavior on the part of the user. For example, an author could give away the right to suppress unauthorized copies just so long as the end user agreed not to create a commercial offshoot.

This argument softened Stallman's resistance to copyright. The copyright law could be seen as just another system waiting to be hacked. This eventually lead to the creation of the *GNU General Public License* (GPL). This license is designed to uphold the four freedoms previously mentioned. The

license permit you to run, distribute, modify and distribute the modification as long as the modified version is also licensed under GPL. Seeing a sticker with the message "Copyleft (L), All Rights Reversed" he decided to call this scheme copyleft using a backward C as a symbol. The GPL is considered Stallman's most important hack.

A lot of widely used and recognised software have been developed by the GNU project. The most important are the *GNU Compiler Collection* (GCC), formerly named *GNU C Compiler*, GNU Emacs, *GNU Debugger* (GDB) and a lot of other Unix tools. A lot of effort by GNU developers went into maintaining and implementing new features into this successful tools. By the early 90s GNU still lacked the most important piece of an operating system, the kernel. Stallman had been looking for a existing kernel to modify. The decision landed on using the Mach micro kernel. The kernel, named Hurd, should be implemented at a set of kernel server for the Mach micro kernel. This work would not commence until 1990, but this would take a long time to do, giving an opening to another kernel which we will see in the next section. In fact to day in 2006 there is still not a stable release of Hurd, but it exist.

4.6 Minix, Linux and Hurd

In January 1991 a graduate student at University of Helsinki had bought himself a PC with a 80386 processor. This came with MS-DOS installed as most PC's at that time. This student, Linus Torvalds, preferred the Unix type of operating systems that he learned about at the university. His apartment was a good distance from the university and student terminals were not always available, so he wanted to run a unix-like OS on his PC. Searching for it he found Minix.

Minix is a small Unix clone made by Andrew S. Tanenbaum for teaching purposes. Tanenbaum is a professor at Vrije University in Amsterdam. He made Minix because AT&T had decided to forbid the teaching of Unix internals. The source code for Minix was published as an appendix to first edition of *Operating Systems: Design and Implementation* in 1987.

At this time the development of the Hurd kernel had started, but no one knew when a runnable version of Hurd would be available. Torvalds installed Minix on his PC and in April 1991 he started to experiment in building an operating system of his own. In the end of August he posted onto the minix newsgroup and stated that: "I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT

clones.". In the beginning of October he posted onto the minx newsgroup again, this time inviting people to experiment with and improve Linux. In this post he also explains his reason for making a new kernel:

```
I can (well, almost) hear you asking yourselves "why?".
Hurd will be out in a year (or two, or next month, who
knows), and I've already got minix.  This is a program
for hackers by a hacker.  I've enjoyed doing it, and
somebody might enjoy looking at it and even modifying
it for their own needs.  It is still small enough to
understand, use and modify, and I'm looking forward
to any comments you might have.
```

It is worth to note that, as mentioned in section 4.3, work to make a free Unix clone and porting it to the 386 processor was already being done at Berkley. Jolitz was working on 386/BSD about the same time as Torvalds was working on Linux. Torvalds would later say that had he known about the availability of 386/BSD he would probably have worked with it rather than starting on his own kernel.

For different reasons Linux attracted a larger following than 386/BSD and its derivatives FreeBSD, NetBSD and later OpenBSD. In 1992 AT&T's *Unix System Laboratories* (USL) filed suite, first against a company named *Berkley Software Design Incorporated* (BSDI) which sold a proprietary offspring of NET/2, BSD/386. Later that year USL refiled the suite against both BSDI and UC Berkley. This created uncertainty around the code from which all the BSD off-springs were based. In 1993 FreeBSD and NetBSD were started based on 386/BSD. In 1995 NetBSD where forked creating OpenBSD as an offshoot.

In the early nineties monolithic kernels, where different components of the kernel like memory management and file systems are all compiled into a single binary, were out of fashion among operating system theorists. One of this theorists was Tanenbaum who made Minix with a micro kernel. In a micro kernel sub components, like memory management, are isolated from a small kernel core. Because discussions about Linux were taking place in a newsgroup devoted to Minix, Tanenbaum posted a news challenging the choice of a monolithic kernel i Linux. This spun of a lengthy debate still available on WWW to day. As Torvalds explained in the previously mentioned e-mail he designed Linux to be a program for hackers by a hacker. Torvalds figured that a monolithic kernel would be easier for hackers to tweak. The choice of a monolithic kernel was also the quickest route to a working kernel. Monolithic kernels is simpler to make in the first versions, but have a tendency to grow into a big, hard to debug, mess.

Unlike GNU's Herd kernel, which was being made with a micro kernel, Linux was available. Simple as Linux still was, it gave promise that it could be made into something great. Torvalds was welcoming to contributions and good at responding to interested people, making contributers feel their efforts were being appreciated and not just thrown away.

By the end of 1991 Torvalds no longer portrayed Linux as hobby for himself. Many who wanted the Linux source code did not have access to the Internet, and therefore could not get it from the FTP site it were distributed. For these people to get Linux sources someone had to copy it to disk and send it. The original homegrown license for Linux did not permit making money on Linux, this included distribution fees. Sending floppy disks cost money, so many developers asked Torvalds to permit a small copying fee. From the 0.12 version released in January 1992 Linux has been licensed under GPL. This license allow a distribution fee, and guarantees that Linux will stay free. In an interview Torvalds said, concerning this decision: "Making Linux GPL'd was definitely the best thing I ever did".

The phase of Linux development grew rapidly from early 1992 and onwards. In early 1992 Orest Zborowski took it on himself to port X windows, the windowing system most common in Unix systems, to Linux. X used a lot of system libraries not implemented in Linux, so the work was more to make Linux fit X than the other way around. The work of porting X expanded the functionality of the Linux kernel, and it exposed many deep bugs.

The next big step was to make a TCP/IP stack for Linux. Because the BSD code was still being disputed in court, the Linux community started from scratch. First Ross Biro made some crude code, which was taken over by Fred van Kempen. van Kempen had a visions to "throw away the old and write it all from the bottom up for a perfect grant vision".

This effort was taking a long time and van Kempen kept his efforts close to his chest. People in the Linux community were getting impatient, van Kempen failed to make some interim code that worked in 80% of the cases. Torvalds sanctioned a parallel coding effort by Alan Cox. Cox had a vision of "make it work first, then make it better". Cox took van Kempen's early code and made it into something useful.

Torvalds chose to include Cox's code into the official version. People started to send networking code to Cox making Cox the semiofficial "networking guy". Torvalds legitimised this by sending networking patches to Cox first before he looked at it. This was the first sign of the "lieutenant model" used for decision making in Linux. This is a hierarchy where code is sent to lieutenants, but where Torvalds have the final say on what is included.

The first stable version of Linux, version 1.0, was released in March 1994. In 1992 Tanenbaum had criticised Linux for being too tightly connected to the 386 processor, making it difficult to port Linux to other hardware platforms. In 1994 a Unix programmer at DEC, John Hall, met Torvalds at a DEC user meeting. Hall was impressed with Linux. Hall later convinced Torvalds to work on porting Linux to DEC's 64bit Alpha processor, with help from DEC. The Linux kernel was re-engineered to make it more portable to different platforms. The most resent Linux kernel release (2.6) support, at least, 17 general purpose architectures. In addition Linux have been ported to a number of embedded systems. It is interesting to see that Linux was criticised by Tanenbaum for being to tied to the x86 architecture and how many architectures Linux supports now.

Many people thinks that Linux is an entire operating system, but it is not. Linux is a kernel. A kernel is a relative small part of what people commonly think an operating system is, like command interpreter, visual display and tools. The kernel is, however, the most important peace of an operating system. A kernel hides all the tricky hardware details of a computer, from programs running on it. Because a lot of the core utilities commonly used in Linux systems to day were made by GNU, FSF claims that Linux should be called GNU/Linux.

To get Linux running on a machine, and getting all the programs you want, was difficult. This gave rise to what is to day called Linux distributions. Distributions conveniently packages a lot of useful software together, and makes it relatively easy to get a Linux based system running. Two identifiable strains of Linux distributions were visible from the early start.

The first strain is based on a community model like Linux is. Example of these are Slackware and Debian, both started in 1993. Debian is the most used distribution, according to the linux counter. The name *Debian* comes from the name of it's founder Ian Murdock and his wife Debra. To stir up some interest for his plans to make a Linux distribution, Murdock posted his intentions on Usenet's `comp.os.linux` and on different Internet sites. This caught Stallman's attention leading to FSF supporting Debian for one year. The Debian project is known, apart for being a good distribution, for the Debian social contract. This contract is the basis of The Open Source Definition which we will look into later.

The other strain of Linux distributions, the commercial distributions, were pioneered by a company calling itself Yggdrasil. Yggdrasil began to sell it's distribution on CD-ROM bundled with non-free software in binary form. In the eyes of FSF this was a sin. Yggdrasil wanted to include the non-free software because they found it useful. Other distributions made with the commercial marked in mind are the European SuSE which started in 1994,

the American RedHat which started in 1993 and the Japanese Turbolinux which started in 1992. SuSE was bought by Novell in 2004, a company that specialize in network operating systems. RedHat now have two distributions. One based on a community model named Fedora, and the other for the business marked named Red Hat Enterprise Linux.

The phase of Linux development grew rapidly during the nineties, and is still strong to day in 2006. The credit file, in the Linux kernel sources, where important contributers are mentioned, have increased from 80 contributer in version 1.0 to 472 in version 2.6.15. The code base for the kernel has also increased manifold, from 176,250 lines of code in version 1.0 to 5,929,913 lines of code in version 2.6.

Up until version 2.2 Linux was a pure monolithic kernel. You had to compile all the features you wanted to include in the kernel and all the device drivers you wanted to have, into one binary. If you added some hardware to your system, you had to compile a new kernel with the device driver of the new hardware included. In version 2.2 *Loadable Kernel Modules* (LKM) was introduced. With LKM, device drivers and extended functionality can be loaded into the kernel during run-time. This makes it easier to extent and test a part of the kernel. It is sort of a middle ground between a monolithic kernel and a micro kernel.

4.7 The rise of Open Source into the main stream

As we have seen it was normal to share code in the infancy of computing , but very few people used computers. Computers were an arcane subject. In the late seventies computers started to come into more widespread use, and it became possible to earn money on software. This created conflict between sharing code and keeping it secret, so you could sell it. This is what Stallman experienced.

Bill Gate's open letter to hobbyist, written at the time, said that the hobby marked lacked good software, and went on to state "Who can afford to do professional work nothing?". As the use of computers grew, so did proprietary software. The "hobby" marked which Microsoft was founded on, would eventually surpass the mainframe and mini computer marked. Proprietary software companies were successful in distributing it's software to the new masses that entered computing during the eighties and nineties.

Stallman had taken the discussion about software into the realm of ethics. He preached that proprietary software was morally wrong. Free software is a question of freedom, you should have freedom to make changes to a

program and give it to your neighbours. FSF's perceived hostility to intellectual property and slightly communist sounding ideology did not sit well with the business world, especially in the US where communism had been seen as the most dangerous threat by the government until the late eighties. The meaning of the word "free" have two meaning in English; free as in "libre" and free as in "gratis". How can you earn money on something that is "gratis"?

In the decade since launching the GNU project, Stallman had built a reputation as an excellent programmer. He had also developed a reputation for being non-compromising both in terms of software design and people management. Ian Murdock, founder of Debian, and Eric Raymond had distanced themselves from FSF because of Stallman's "micro-management" style. Eric Raymond had up until 1992 contributed significantly to GNU Emacs, but distanced himself for the same reason. In 1996 FSF experienced a full-scale staff defection, blamed in large parts on Stallman.

> Brian Youmans, a current FSF staffer hired by Salus in the wake of the resignations, recalls the scene: "At one point, Peter [Salus] was the only staff member working in the office."

(Williams 2002)

Hackers like Linus Torvalds and others in the Linux crowd did not share this moral view on software. Torvalds do not find proprietary software morally wrong. He started Linux development for the fun of it, not to fight proprietary software. In an interview Torvalds said:

> I'm generally a very pragmatic person: that which works, works. When it comes to software, I _much_ prefer free software, because I have very seldom seen a program that has worked well enough for my needs, and having sources available can be a life-saver.

(Yamagata 1997)

There were growing dissent in the free software community towards FSF's strongly moral stance. There were a large contingent of developers within the free software community having a more pragmatic view on software, not only Linus Torvalds. The free BSD Unixes were released under a license with minimal restriction, permitting proprietary derivatives from the code base. This crowd believes in the freedom to build great software with minimal restrictions.

The previously mentioned Eric Raymond encountered Linux in late 1993 and what he saw came as a shook to him. He assumed that hacker amateurs could not muster the resources to produce a multitasking operating system. He involved himself in the kernel development, and pondered upon what made Linux development work so well.

Brook's Law predicts that as the number of programmers N rises, work performed scales as N, but complexity and vulnerability to bugs rises at N^2. So if a project is delayed it will only be more delayed if you add programmers. Raymond was determined to discover how the Linux community had avoided the N^2 effect.

After three years of participation he developed a theory. He tested the theory on the procmail project. Based on his experience with Linux and procmail he wrote *The Cathedral and the Bazaar*. In 1997 he presented this essay on a Linux congress in Germany and a Perl conference in USA, with standing ovation from the audience. The news about this essay spread like fire on the net.

Netscape, a pioneering company in web technology, had been targeted for destruction by Microsoft. Microsoft used it's near monopoly of the desktop by including Internet Explorer in it's operating system offerings. Netscape feared that if Internet Explorer achieved marked dominance, Microsoft would be able to bend the web protocols away from open standards and into Microsoft's own proprietary standards, which only Microsoft's servers could serve.

In January 1998 Netscape announced that it would release the source of the Netscape browser. The content of *The Cathedral and the Bazaar* was a major influence in this decision. It took several month after the announcement before the code was released. The code for the Netscape browser was a hard to understand patchwork, with inadequate documentation, developed under tight business deadlines. The code badly needed reorganisation, which took several months.

Eric Raymond saw this as an demonstration case for the principles he laid out in *The Cathedral and the Bazaar*. He hoped that if the source code release was successful the hacker culture, and thereby free software, would raise out of it's ghetto and into the mainstream, if not he feared that it would confirm business manager's assumption that free software were not commercially viable.

Raymond offered his help to Netscape in developing the license for the software and working out a strategy. While he was meeting with Netscape he also met a number of key people in the free software community, like Li-

nus Torvalds and Bruce Perens. During this meetings a strategy for getting free software into the mainstream was made. In February 1998 Eric Raymond and Bruce Perens founded the *Open Source Initiative* (OSI). Deciding on using the term *open source* to avoid the "libre" vs "gratis" confusion associated with the term "free", Raymond and his supporters set out on a marketing campaign outlined in his book (Raymond 2001).

I March Tim O'Reilly, founder of a publishing company specialising in software related books and an early supporter of Raymond's initiative, gathered a number of key developers living at the west-coast in the US. This was set in place to get support for OSI. The invitation list included Torvalds, Larry Wall (creator of Perl), Eric Allman (creator of sendmail) and Paul Vixie (creator of *Berkley Interned Naming Daemon* (BIND), the most used DNS server.). This was later called *The Free Software Summit*. In this meeting it was agreed upon to use the term open source with a 9 to 15 vote, according to O'Reilly (Williams 2002). Stallman was not invited to this meeting which would create controversy later.

The OSI marketing campaign can be said to have been a success. It created a lot of attention in the US media. Other companies began to announce that it would involve themselves in open source. Oracle, a big database vendor, said it would port it's database to Linux. IBM involved themselves in the Apache project. Apache is the web server most frequently used on the net. IBM used Apache in it's WebSphere product and contributed back to Apache, even if the Apache License did not require it.

The open sourcing of Netscape did not become a success, but it spun of the Mozilla Foundation. The Netscape source was still too hard to understand and had to many interlocking dependencies. It was decided to build a new layout engine for the browser from scratch. The layout engine turns HTML into what is rendered in the browser. This engine was built using a *Component Object Model* (COM) framework and given the name Gecko. Gecko made the development of the web browser highly modular. The Mozilla Foundation have produced the Firefox web browser and Thunderbird e-mail client, which is increasing in popularity. The open source initiative gained so much momentum that the early failures of Netscape to open source it's browser did not matter.

Bruce Perens resigned from OSI after one year regretting that OSI had positioned itself in opposition to FSF. In a e-mail from February 1999 he wrote:

```
Most hackers know that Free Software and Open Source
are just two words for the same thing.  Unfortunately,
though, Open Source has de-emphasized the importance
of the freedoms involved in Free Software.  It's time
```

```
for us to fix that.  We must make it clear to the world
that those freedoms are still important, and that software
such as Linux would not be around without them.
```

(Perens 1999)

Stallman considered briefly to adopt the term open source, but concluded that: Open source, while helpful in communicating the technical advantages of free software, also encouraged speakers to soft-pedal the issue of software freedom.

The pragmatic view represented by OSI, emphasises the technical and efficiency advantages and represents the Linux community as an example of an efficient method. The moral view represented by FSF, emphasises freedom and the right to change and distribute software. This differences have not been settled, so the term FLOSS have been invented to include both the free software and the open source crowd. This is the term I will be using in this paper.

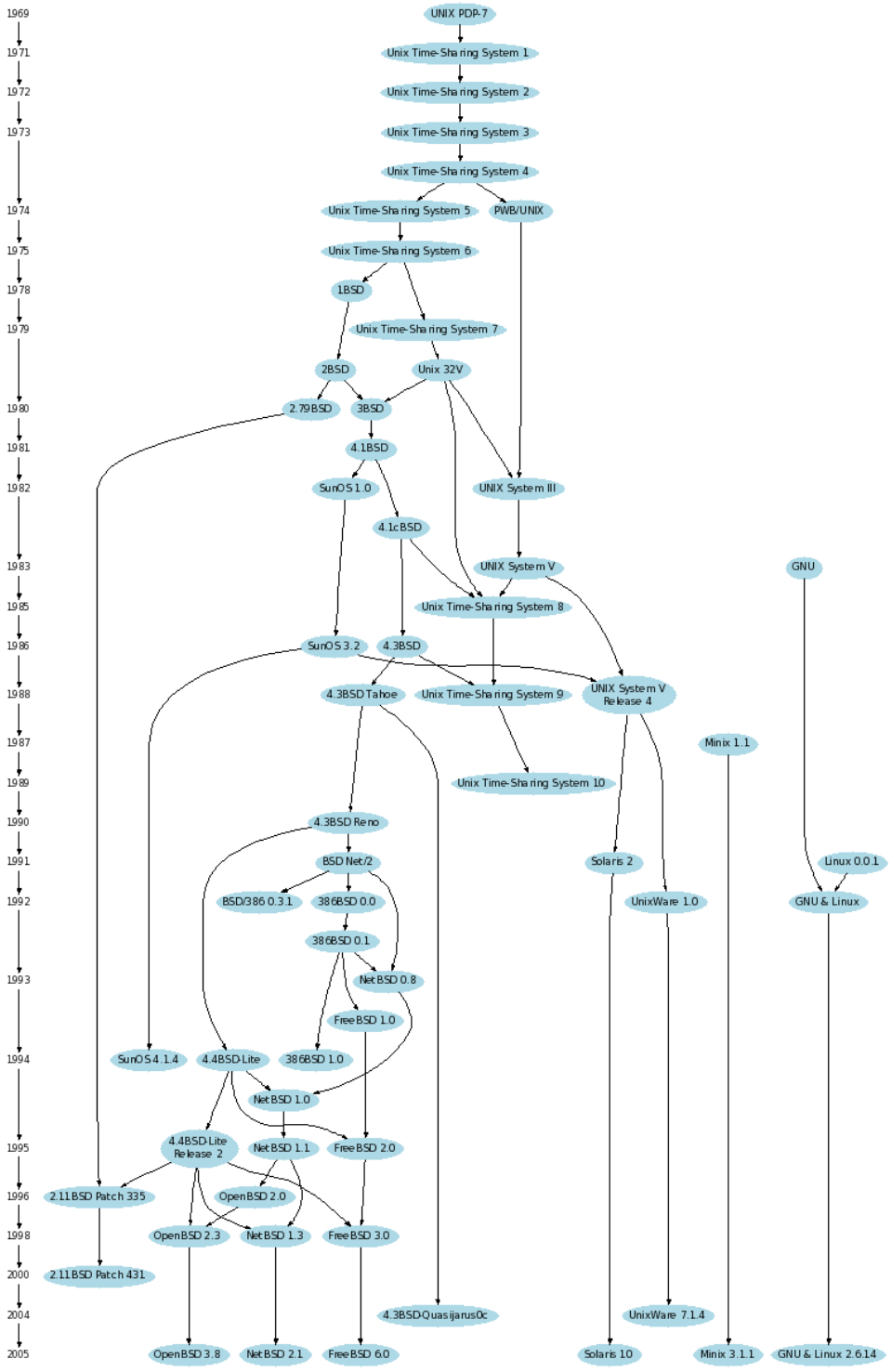

Figure 4.1: Evolution of Unix

Chapter 5

FLOSS - How does it work?

Many have marveled at how it is possible to make software as complex as operating systems without handing out paychecks and having strong management. How is it possible to motivate people to do hard and complex work without paying them? And how it is possible to get, up to as many as, hundreds of people to work efficiently together towards a common goal, without strong management? This are question I will explore in this chapter.

To understand FLOSS some knowledge about the philosophise and values common among FLOSS participants and FLOSS leaders are necessary. Then we will look into what motivates FLOSS developers and then on how project in FLOSS are governed and we will look at the FLOSS license. When the general introduction to FLOSS is finished I will present challenges and constraints in the FLOSS development model. Last I will move on to give a presentation of FLOSS in the context of developing countries.

5.1 Philosophy and values

People working with FLOSS are not a unified group. There are people of a multitude of political, religious and moral views working with FLOSS. An effort to give a short and specific explanation of FLOSS philosophies and values necessarily has to be somewhat imprecise. One common factor among all in the FLOSS community has to be the view that source sharing is a good thing, for whatever reason. There are organisations who seeks to represent the FLOSS community and some empirical research exploring motivation among FLOSS participants have been done

The organisation FSF are seeking to represent and influence the FLOSS community, and has a lot of expressed philosophies and values, with a strong moral stance. OSI has less focus on moral philosophy and values and emphasises the economic and engineering aspects of FLOSS.

Many important figures in the FLOSS community, like Linus Torvalds, Richard Stallman and Eric Raymond, identify themselves, and are being identified by others, as hackers. This show that the *hacker* term is important in understanding the FLOSS community. So in our quest to identify common philosophy and values in the FLOSS community it is important to understand the hacker ethics.

You don't have to be a hacker to participate in FLOSS. To contribute code, however, you have to like programming, or perhaps being paid to program. The hacker community, judging by the Hacker Survey (Lakhain, Wolf, and Bates 2002), is a core part of the FLOSS community. To the statement "Hackers are a primary community with which I identify" 41.5% strongly agreed and 42% somewhat agreed. FLOSS has a strong historical heritage from the hacker community, and the hacker community is still dominating the FLOSS scene.

This is in the process of changing as FLOSS has become more main stream. People are hired to work with FLOSS. 30% of the respondents to the Hacker Survey gave "My contribution creates specific functionality in the code needed for my work" as one of the three top reasons for contributing code to a project. There are businesses experimenting with different business models to profit from FLOSS. IBM HP, RedHat and MySQL AB are example of companies participating in FLOSS.

I will in the following sections more closely describe the hacker ethic. Then I will look into the more pragmatic and technical philosophies in the FLOSS community. Last I will look into the moral philosophies represented by the four freedoms promoted by FSF.

5.1.1 Hacker ethic

Unlike the public image of the stereotype hacker, a person identifying himself as a hacker is usually not trying to break into private networks. Even if the persons doing this frequently call themselves hackers, and is called hackers by the public, this is not the type of hackers I am going to describe here. The type of hackers I am referring to is the type of hackers defined in the *Jargon File*. According to the Jargon File, maintained by Eric Raymond , a hacker is (Raymond):

1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary.
2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.
3. A person capable of appreciating hack value.
4. A person who is good at programming quickly.
5. An expert at a particular program, or one who frequently does work using it or on it; as in "a Unix hacker". (Definitions 1 through 5 are correlated, and people who fit them congregate.)
6. An expert or enthusiast of any kind. One might be an astronomy hacker, for example.
7. One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations.

The interest in exploring programmable systems and the intellectual challenge of creatively overcoming limitation can lead to the “dark side” of hackerdom. The “dark side” meaning breaking into computer systems to make damage, and/or making viruses, worms and other nasty harmful programs. A “dark side” hacker is called a *cracker* in the Jargon File.

The same Jargon File also define the ethics commonly adhered to by hackers:

1. The belief that information-sharing is a powerful positive good, and that it is an ethical duty of hackers to share their expertise by writing open-source code and facilitating access to information and to computing resources wherever possible.
2. The belief that system-cracking for fun and exploration is ethically OK as long as the cracker commits no theft, vandalism, or breach of confidentiality.

Information sharing is different from sharing of material good in the fundamental way that the one who give information do not loose the information he gives. This is illustrated by this African saying:

> Two little boys exchanges toys, both went away with one toy each. Two wise men exchanged ideas, both went away with two ideas each.

Through information sharing you get access to the work and ideas of numerous others. By contributing just one idea, millions of others can benefit from it without you loosing anything, more likely you earn respect from it. This is a clear parallel to the academic community, where the open exchange of ideas is essential, but the hacker community has less obstacles to participation. The hacker community is informal, while the structure in the academic community is clearly more rigid, with positions and titles. The second point in the hacker ethic, the one about system-cracking for fun, is more disputed.

Hackers have a belief that there are a near to unlimited number of challenging and interesting problems to work on. There is a thrill in solving intellectually challenging problems, though a thrill that require hard work. If the problem is solved in a elegant and creative way it is even better. Because there are so many interesting problems to work on no problem should have to be solved twice. If a solution is kept secret, the problem has to be solved again for others to benefit from it. Reinventing the wheel should be avoided whenever possible.

The hacker culture is strongly meritocratic. You can not give your self the honour of calling yourself a hacker, but based on your merit this title might be awarded to you by other hackers. The measure of prestige in the hacker culture is the code you produce, the software documentation you write, the tutorial for a computer language you make, or other signs of good craftsmanship in software related tasks you show. The culture do not judge according to gender, race, title, but it do judge according to your ability to contribute. Bragging about your abilities would usually not help to improve your merit, let your work brag for you.

Within the hacker community there is disagreement to whether *all* information/code should be free. The OSI camp emphasises that it is preferable with free information because it gives better results. In a way you could say that we benefit most from information when it is distributed freely, but you can keep it to yourself if you want to. The FSF camp says that it is morally wrong to deprive your neighbor of information. This views we will look closer into in the following two sections.

5.1.2 Pragmatism

OSI, Eric Raymond and Linus Torvalds represents the more pragmatically inclined contingent of the FLOSS community. The emphasis is on the joy of programming, and on how good the FLOSS model is to create good stable software. FLOSS gives you the ability to look into other peoples code, so

you can learn from and improve it. FLOSS has so many sensible engineering advantages that it is not necessary to go into the realm of ethics. To keep information to yourself is not morally wrong, but it would be much more helpful if the information is allowed to be used and built upon freely.

The pragmatic reasons sounds better to the ears of a business manager. If your business is in need of software, where the software itself is not meant for sale, like an e-commerce application, you basically have three alternatives. You can buy a proprietary solution from a vendor, you can build your own or you can base it on FLOSS solutions. For businesses that do not have it's profit from software sale, the large majority, but depend on software to run the business it makes sense to share the cost of developing the necessary software through sponsoring a FLOSS project. Like the previously mentioned SHARE from the fifties shared code, because hardware was seen as the main business.

A majority of software developers do not work on software meant for sales. The majority of developers develop software because a company need it as a service to support it's revenue generating enterprise. To look at software production though the spectacles of industrial production is misleading in this cases. A developer is not a point in an assembly line, but is more like a doctor asking where it hurts and working to remedy that. Having access to a large tool-set of FLOSS applications makes this much easier and saves the company money.

Many other pragmatic reasons can be given, but the point is that it is many different reasons for the resent interest in FLOSS. For sure most of this interest is based on pragmatic reasons, few of them particularly idealistic. There are arguments for using FLOSS software and sponsoring FLOSS development based entirely on economic considerations.

5.1.3 Moralism

Parts of the FLOSS movement represented by FSF have elevated the concept of free sharing of information into the realm of ethics. The key elements in the ethics promoted by FSF is outlined in the four freedoms listed in section 4.5. By keeping information to your self you are hoarding information. Restricting access to information is to deprive others of the freedom to use and build upon this information.

The grand old man of FSF, Richard Stallman, in his article *Why Software Should Not Have Owners* (Stallman 2002) presents key points in the ideology behind free software. Copyright fitted well with the printing press because

it restricted only the mass producers, but to restrict copying and editing of digital information restricts the right of the individual user. Stallman argue that the authors have no natural right over what they write, that is why copyright are limited in time. The concept of copyright exists to give the authors some economic gain from their work in order to promote development. Before the computer, copyright only restricted mass producers, but as distribution of information electronically is much simpler than distributing by paper copyright do not restrict only mass producers, but individual users. FSF has made use of copyright in the previously mentioned copyleft scheme. This scheme ensures that the software stays free by requiring modified versions to stay under the same license.

The previously mentioned pragmatic advantages FLOSS gives are happy consequences of freeing information, but it is fundamentally a moral question. In the Hacker Survey 34,2% gave "Code should be open" as one of the top three motivations for FLOSS participation. The majority of FLOSS participants are participating because of pragmatic reasons, but still a substantial part are idealistically motivated. The FSF with it's moral message kept to the ideals of free software in the eighties and nineties at a time when few believed that free sharing of source code would be anything more than a small niche. Even if the pragmatic reasons for FLOSS are dominating, the moral message is important for bringing the FLOSS concept to life.

5.2 Development practices

In his book *The Cathedral and the Bazaar* (Raymond 2001), Eric S. Raymond divides between two different paradigms of software development; the Cathedral and the Bazaar. The cathedral represents the hierarchical and ordered organization of the big software companies. The bazaar on the other hand represents the many independent actors in the open source community. This comparison gives a good picture of the differences between companies and the open source community, but is not all together true. All companies is not necessarily strongly hierarchically organized, and the Open Source community have many organizations that gives structure to the development. I will in this section look closer at the *Bazaar* model, with which I mean the distributed innovation done in open source projects.

To build usable software is a extraordinary complex process. It is hard to predict the time and money required to build an application. The computer software history is full of examples where project have blown the budged and gone severely over schedule, and sometimes never finished. This complexity exist for both proprietary developed software and software devel-

oped by an open source project. This complexity however is handled in different ways. In a business setting complexity is often handled through distribution of labor. The scheduling of tasks is most often handled in a top-down fashion by management. Because different part of an application often have complex dependencies, it is exceedingly difficult to make good decisions about how labor should be distributed. There is no question about the need to distribute software development, the task of making large computer software is too big for anybody to do alone. Unlike when an architect is drawing a building it is not possible for a software architect, or a group of software architects, to make a drawing, or plan, of the software and hand it down to the workers and expect that the software will become as he imagined. The field of software development is simply a to complex and lucid a thing, to accurately pin down in general terms. The *bazaar* and the *cathedral* are illustrations on how labor are distributed.

In an FLOSS setting division of labor is done in a quite different manner. Production in a FLOSS project is based on voluntary labor. A company hands out reward in form of payment and can threaten to fire an employee if he does not fulfill the requirements laid on him. Because of this the company can make the employee perform a task that he finds boring. This is not the case with voluntary labor. In an FLOSS setting you have the chance to choose what you want to contribute. There exist a multitude of project that you can choose to contribute to, and if you find a need in an area you are interested in you can start your own project to fulfill that need.

This motive to participate in or start a project in a area you are interested in is labeled with the phrase “to scratch a personal itch”. In this scenario the developer is both the user of the software and the developer. Close communication between users and developers is critical to the successful development of software with non-trivial and/or uncertain requirements. When the user and developer is merged into one, this communication is as perfect as it can get. Participants in a FLOSS setting are user-programmers, with varying degree of emphasis on user or programmer. There are a varying degree of sophistication and involvement by the user-programmer. The different degrees can be exemplified by the following phrase: “All can make suggestions, many can send bug reports, some can make patches and a few can make major contribution.”

FLOSS developers being motivated by interest might lead you to think that a lot of boring task will not be done. Among programmers the tasks of user documentation and visual design is often not found interesting, but participation in FLOSS is not limited to programmers. As an example I can mention freecol http://www.freecol.org/. Freecol is an FLOSS implementation of a computer game called Colonization. In this project a music composer, two

graphics designer and a writer is mentioned on the Credits page. FLOSS gives people with different interest a chance to do what they like, and at the same time give them a chance to share it with the world. It seems that the way of organizing complex task through individuals choosing tasks is a viable alternative. It is true, however, that proprietary software usually are more pleasing to the eye. After all like many other commercial products, proprietary software are in large part dependent on the wrapping to sell.

There is a danger that the code you develop will not be accepted into the project. When this happen one choice FLOSS provides you is the possibility to fork the original project, that is to take the source code and distribute your own version. This is a danger taken seriously by the FLOSS community. This do not happen frequently because among other reasons you have to build a community around your distribution and because you can not take the trade mark. If the time you spent on making code that are not accepted was measured in money, as it would be in a corporate setting, it would be considered a great waste.

Because FLOSS project often depend on contributions from many developers and because FLOSS do not suffer from the tight schedules that are common in corporate settings, you both have the incentive and time to give proper structure and documentation to the code being developed. For new people to be able participate it is important that the entry level to understand the code is not prohibitively high.

When Netscape Communication decided to release the code for their browser they had to restructure the code. Their browser was developed in a corporate setting with a tight schedule, but when they wanted to build a community of voluntary developer around the browser Netscape understood that it had to represent the code in a more communicative way. Still after restructuring the code it was found to difficult to build upon. The Mozilla project which the FLOSS project that spun off from the Netscape source release was named, decided to build the browser based on the FLOSS Gecko layout engine. This allowed for a much more modular design of the code, which made code contribution much easier.

5.3 Motivation

According to Steven Weber in his book *The Success of Open Source* (Weber 2004) there are two principal forces at work in FLOSS, individual motivation and governance. He is actually saying a bit more than that, but this is my nutshell interpretation of what he says. I think he has a good point so here I

will take a brief look at what motivates people to participate in FLOSS and in the next section I will look at governance structures in FLOSS projects.

Because participation in an open source project is basically voluntary, individual motivations for participation is a critical aspect of the open source community. At first it can seem strange that people would spend countless hours in front of their computer doing hard, time consuming and sometimes even frustrating work. My experience as a programmer is that at many stages in development you face frustratingly hard challenges. This work is often done with small possibilities of ever making money from the effort. Is this done with the altruistic motivation of giving others useful software, or are there other motives behind this?

Although I said that programming challenges can be frustrating at times, to overcome this challenges is a relieving and fulfilling sensation. My first guess therefore of why people choose to contribute is because we as humans have a longing to create. We are not happy with only food and comfort. In our western world food and comfort is a commodity, and there is a limit to how much money you need to make a good living. If your choice stood between doing creative and interesting work for free and repetitive boring work for money, what would you choose? When you have enough money to pay your bills why would you choose a boring job for its payment? That is if you don't see money as an end in it self.

Research on motivation and performance shows that external rewards in the from of bonuses etc. for performance will reduce creativity and the intrinsic interest in the task, if the focus becomes directed to the reward. Not only do it not help much in creative task with extrinsic rewards, it actually often lead to poorer performance. If you are rewarded for a task the task itself becomes only a mean to an end and therefore the interest in the task is reduced. This is illustrated by this joke (Kohn 1987):

> An elderly man, harassed by the taunts of neighborhood children, finally devises a scheme. He offered to pay each child a dollar if they would all return Tuesday and yell their insults again. They did so eagerly and received the money, but he told them he could only pay 25 cents on Wednesday. When they returned, insulted him again and collected their quarters, he informed them that Thursday's rate would be just a penny. "Forget it," they said - and never taunted him again.

Rewards and punishment can have a short term effect on modifying behavior. For mindless repetitive tasks, which have little intrinsic motivation, extrinsic rewards will increase performance. This is because the task

is seen as having little value in it selves. For tasks where the quality, understanding, learning and creativity is more important than the quantity (like computer programming) intrinsic motivation is most important. If an extrinsic reward is seen as confirming the quality of your work, and your competence, it can be helpful. If the extrinsic reward is seen as way to control your behaviour it is counterproductive.

The action of handing out your work through a license like the GPL might seem altruistic, but I don't think altruism is the primary driving force behind FLOSS. If you contribute because of an urge to create does not mean that you have the good of others in mind. This urge to create you can see in other endeavours like art. I see some similarities in what motivates people to do art and what motivates people to hand out their code to the public. Both comes from an urge to create and perhaps change the world just a little. This urges are not necessarily altruistic.

In his book Weber propose a scheme that captures six kind of motivations seen among FLOSS developers.

1. Art and beauty
2. Job as vocation
3. The joint enemy
4. Ego boosting
5. Reputation
6. Identity and belief system

I will not elaborate too much on this points. This points needs more explanation, however. *Art an beauty* I have already mentioned which relates to *Job as vocation*. The work of programming is more than a job to put food on the table. *The Joint enemy* with Microsoft as the obvious villain, which is a primary motive of only a few according to a survey made by Boston Consulting Group (Weber 2004, page 139). This is important none the less. As a teacher in history I once had frequently said, an outward enemy creates inward unity. Microsoft, as an enemy, is something FLOSS developers can identify as being in opposition to. *Ego boosting* in FLOSS does not mean boasting about your excellence as a programmer. Such behavior is generally not approved. The point of ego boosting in FLOSS is that your work boast for you. The point of *reputation* is both because a good reputation give an external measure on the quality of your work and because it help

others to determine your quality as a programmer. To have a good reputation helps in a competitive job marked. *Identity and belief system* refers to a common identity and common values among FLOSS developers.

The Hacker Survey segments the hacker community according to motivation. The respondents are segmented into four groups:

Believers 33% of the respondents. Participated in FLOSS because they believe source code should be open.

Fun seekers 25% of the respondents. Do it for non-work need and intellectual stimulation.

Professionals 21% of the respondents. Do it for work need and professional status.

Skill Enhancers 21% of the respondents. Do it to improve coding and other related skills.

5.4 Governance

Although the FLOSS community is very individualised it is not enough to explain FLOSS by individual motivations. Individual motivations is the cornerstone of the community, but efforts needs to be coordinated. Unlike many genre of art you need to work closely with other practitioners of the coding "art". It is simply to much work to do for one individual alone. For small projects with just a few developers, which is the majority of FLOSS projects, the question of governance and management is not very important. It is only for projects of some size that the previously mentioned N^2 effect (Brook's Law) becomes relevant.

Maybe the most important aspect that help to frame the FLOSS community is the license. This is mentioned in section 5.5. But other factors are important in how FLOSS is governed. A basic value of the community is that your work speaks for you. It is at its core a meritocracy. According to the survey previously mentioned 48.4% of the respondents mentioned among the three most important things a leader should do, that the leader should "Create initial code base". 34.4% answered "Contribute code" (Lakhain et al. 2002). This shows that the ability of a leader to create code is what legitimise him most as a leader.

Linux is an example of such. Linus Torvalds started the development of Linux as a post graduate student. He initially released a small, but working kernel. Many others later joined the project, but Torvalds still have the

final saying in what is going to be incorporated in the official releases of the Linux kernel. Torvalds gained his initial leadership role because he made the first release. He kept his leadership role and avoided forking because of a combination of responsiveness and charismatic leadership. Because he responded to patches and suggestions contributors felt that their work were appreciated. After the Linux kernel and the Linux community grew Torvalds had problems responding to the community, because it was to much for one man to do. This is often called the "Linus don't scale" problem. This was solved by giving the responsibility for different sub systems of the kernel to *lieutenants*, who could approve patches. This made the Linux project into a hierarchical ordered organisation.

This story about Linux points to tree important organising factors common is FLOSS. That is the right to submit code into an official release, charismatic leadership and meritocracy. Because it is difficult to build a community of developers around a forked version of code, the ability to decide what is to be included in an official release gives power. The charismatic leadership of Torvalds is important for Linux, but other forms of organisation exists. In the Apache Foundation a board is elected by the foundation's members. In that kind of an organisation charismatic leadership is less important. In all cases the time, effort and skill you put into making working code is what most give you the power to influence an FLOSS project.

Even if the management in FLOSS is not strong and coercive and most often do not have the power to punish and reward, it is still important with management. As we have seen punishment and reward is a poor motivator for creative work, so the more informal structures in FLOSS is an asset. Successful management is FLOSS is more in the spirit of this saying of Jesus (Matt 20:25-26):

> You know that the rules of the Gentiles lord it over them, and their high officials exercise authority over them. Not so with you. Instead, whoever wants to become great among you must be your servant, and whoever wants to be first must be your slave.

Management in FLOSS must be with a serving attitude. If a developer do not like the way he is treated he can just leave, and there is also the possibility of forking the code. This to not imply toothless leadership, you must be able to give direction and convince others about your decisions.

Weber tries to capture what FLOSS participants do and how project are managed in the following eight point. This points are most relevant for the projects which are based on the bazaar model, or as I prefer to call it a

community model. This is because community gives more sense of identity and structure than the word bazaar.

1. **Make it interesting and make sure it happens.** FLOSS developers choose to participate and chooses which tasks they want to contribute to. Many of the tasks that needs to be done to make good software is uninteresting. What is thought of as interesting or uninteresting will vary among contributors. With a community of some size associated with the project, a project leader can hope that someone will find a particular task interesting or valuable as a learning experience. To make sure an uninteresting piece of work get done, the project leader engage the community and encourages someone to step up to the task. Doing such tasks increases a contributor's standing in the community.

   The contributors want their work to be appreciated by seeing their work getting into actual use. Projects that looks like it will generate significant products will attract more contributors. If the project can offer interesting programming challenges it is even better. The contributors don't want to waste their effort on dead-end projects.

2. **Scratch an itch.** The itch a FLOSS developer can scratch is the problems he faces in his immediate environment. Most software code is written for in-house use, estimated to about 75%, and are not written for sale. It is much more cost efficient for a business to contribute to an existing project to make it useful for a specific in-house use, than it is to build the same from scratch. Besides, this releases the business from maintaining a lot of in-house built code. For hobby programmers there are other itches. Linus Torvalds wanted to have an operating system similar to the one they had at the university, at home. People contributed to Emacs because they wanted the editor to do more useful tings.

3. **Minimise how many times you have to reinvent the wheel.** To write good quality code is hard and time consuming work, so FLOSS developers tries to avoid writing code for problems that are already solved, whenever this is possible. Volunteer FLOSS programmers have an extra strong incentive to avoid this, since they are not compensated for their time. Within proprietary settings there is constantly a need to reinvent the wheel, since the code is not available to the competitors.

4. **Solve problems through parallel work process whenever possible.** In the conventional engineering archetype there is one or a few architects that are responsible for deciding what need do be done, and to delegate sub-tasks to the hierarchy beneath the architects. The architects defines the problems that needs to be solved. On a FLOSS project the

project leader can set the effective definition of a problem that need to be solved, in other cases the definition can be given by a desired feature request, suggestions or by the discovery of bugs. There can potentially be many ways to solve a problem. By attracting a number of developers to work on this problem different routes to a solution can be explored. No central authority decides which routes should be explored or who should explore them.

There can also be problems that are not tied to a specific project. One example is the problem of making and displaying dynamically created web content. This have given rise to many different web development frameworks. Different solutions to this problem have given rise to different projects.

5. **Leverage the law of large numbers.** Even a moderately complex program has a functionally infinite number of paths through the code. To discover bugs in the code as many as possible of this paths should be tested. The developers of this program have preconceived ideas of how this program are to be used. The consequence of this is that only a tiny portion of the possible uses of the software will get tested by a few developers. By submitting the code to massive peer review, and by letting many people use the software, more bugs will get uncovered, and the fix will be obvious to someone. "Given enough eyeballs, all bugs are shallow" (Raymond 2001) is a phrase used to describe this. By having different people doing different things with the software more bugs will get uncovered. The bug can then be communicated to an audience of possible fixers. For one of these the fix will be obvious. Lastly the fix have to be incorporated into the main code base.

6. **Document what you do.** Documentation is important in both proprietary and FLOSS settings. It is difficult and time consuming to deduce how a program, or piece of code, function by simply reading the code. In proprietary settings developer documentation can often be substituted by face-to-face communication because of the proximity of the developers. Proprietary software can be developed in a distributed manned within a large enterprise, and FLOSS can be developed by a tightly knit group without contribution from the outside. Distributed development is more common in the FLOSS setting, and is facilitated by the openness of the code and the liberal licences for distribution. For distributed development good documentation is important for the different contributors to get up to speed on what can and should be done. To attract volunteer user-programmers into contributing to a project good developer documentation is a way to lower the entry barrier. Developer documentation transfers the knowledge of the

software author across time and space. Good user documentation is equally important for FLOSS and proprietary software, if they want their software to be used.

7. **Release early and release often.** FLOSS project built on the community model can harvest more contribution by early and frequent releases. In this way contributors can see the result of their contribution quickly and bugs can be discovered more quickly. If the number of new features contributed to a project gets too high it can overload the selection mechanism, selecting which features to include in the official release. Each new feature included can contain new bugs, so the number of new features have to be balanced against the desired stability of the code. This is why many project have a development and a stable release.

8. **Talk a lot.** Interaction is the mean by which social systems are changed and replicated over time and space, to recapitulate a central concept from structuration theory. In community based FLOSS projects the activity on e-mail lists, discussion forums and other mediums for communication is fundamental in creating communal identity around a project. A common misconception about the community based FLOSS process is that it involves like-minded geeks who cooperate and agree with each other based on technical grounds. The tone of conversation if often far from calm and polite, and disagreements, frustrations and criticism are openly vented. Discussions about technical problems is generally grounded in a belief that there exist a technical solution. However, the solution is not always obvious and there can be many contending solution.

5.5 Property, Copyright and Licenses

Licenses acts in the FLOSS community almost like a constitution. It gives the fundamental "rules of the game". It is through licenses the basic principles of FLOSS is maintained. There exist many different licenses that claims to be FLOSS. In an effort to state some formal principles on what constitute FLOSS the OSI www.opensource.org have made the *Open Source Definition*. Based on this definition OSI offers certification of licenses to help people distinguish between what is and what is not a FLOSS license.

Those licenses that are approved by OSI can be divided into two categories. The first is the GPL style license and the second is the BSD style license. GPL is an acronym for General Public License and is one of the most used licenses. The GPL license was made by FSF in order to make free software

possible within the current copyright regime. In the GPL license the author gives explicit rights to copy, modify and distribute software. In addition it requires software derived from GPL code to be licensed with GPL. This requirement is often called the "viral" clause, particularly by its opponents.

The purpose of the "viral" clause is to stop anybody from taking free software, modify it and release it under a proprietary license. When Richard Stallman was working on a Lisp interpreter a company called Symbolics asked to use his interpreter. Stallman gave them a public domain version of his interpreter. Symbolics extended the Lisp interpreter. When later Stallman wanted to access this extensions he was refused. The primary reason behind the GPL is to prevent this kind of behaviour. FSF designed the GPL to make sure that once software is made free it stays free.

The BSD kind of licenses is similar to the GPL style license, it allows for modification and distribution, except it does not have the "viral" clause. The BSD license in it self is made by Berkeley University and is very permissive. It basically says that you can do what you want with it, but use it at your own risk.

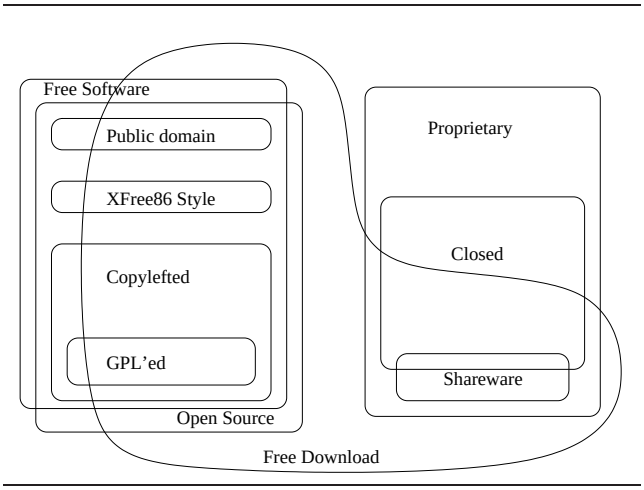

Figure 5.1: Diagram by Chao-Kuei that explains different categories of software licenses

Weber (2004) interprets this to mean that the difference between this two styles of licenses are based on different interest in the software. The GPL style license are based on a long term interest in the software, it's intention is to keep the software free during the whole life cycle of the software. It is based on a philosophy that software should be free (as in free speech) and therefor the license must secure that the software stays free. The BSD style licenses is made by an university which is often not interested in the

software on a long term basis. After a research project is finished the researchers wants to move on to other projects. The BSD style license is also preferred by many in the business world because it makes it legal to make proprietary derivation of the work.

A variation on the GPL style license is whether the license allows proprietary software to link with the licensed software. This allows proprietary software to use the licensed software through calling it's functions/methods, extending its classes, implementing its interfaces and similar methods of linking to the software. Extension to the software itself must be released under the same license. The original GPL do not allow this, but the *Lesser General Public License* (LGPL) do. The intension is to spread the use of the licensed software, even as part of proprietary software, but at the same time to ensure that the software itself stays free. GNU calls this license *Lesser* because it only encourages it to be used for libraries, databases etc., where there already exists proprietary alternatives. For a figure on the different categories of software licenses see figure 5.1.

5.6 Challenges and constrains of FLOSS

In this section I will identify some tentative challenges and constraints of the FLOSS process. This I have identified from reading about FLOSS and from talking to different people. I do not have any references on this and it is only my opinion, based on direct and vicarious experiences.

One of the greater challenges meeting FLOSS at this time is the limited focus given the end user experience, or wrapping if you will. For many commercial products the wrapping is one of the most important elements, if not the most, that need to be in place in order to sell. The wrapping is the first thing a user sees, and first impression is important. Many hacker dislike "bells and whistles" which is functionally unnecessary but visual pleasing. Hackers often prefer the command line, and some even look at GUIs with a certain antipathy. I do not think this is a limitation in the FLOSS process. FLOSS have just recently come to a level of maturity where it is meaningful to focus on end user experience. Now that there are tools available to work with image processing and good libraries to build GUIs, and more people are aware of FLOSS, people who have an eye for esthetic qualities can be attracted to contribute.

Everybody, even FLOSS developers, needs to earn a living. Before anybody is willing to contribute for no monetary benefit they have to have food on the table and be able to pay their bills. When a developer has to prioritise

his time, voluntary labor will most likely loose to labor the developer are paid to do. Some FLOSS developers are payed for their work, this frees the developer's time, but most developers are not. Getting large contributions from companies, or being hired by a company to work on a FLOSS project, can free a developer's time. This can unfortunately create conflict of interest between commercial interests and the interest of non-commercial users or user-developers. Projects run into the danger of being co-opted by commercial interests.

Free-riding, where someone use the software but never contribute back, is not a problem if the free-rider is an individual user. To have many users, contributing or not, is an asset to a FLOSS project. Except for download bandwidth, it cost no more for a project to have ten thousand compared to one million users. A large user base creates a positive network effect where the users do marketing for the software, and they do support by helping their friends. For commercial user making good money from FLOSS the question of free-riding is different. This is the primary reason for the "viral" clause in the GLP license.

A majority of FLOSS projects are small and a lot of effort is "waisted" on marginal projects. Because this is done voluntarily it has no monetary cost. If the time spent on coding a piece of code that never came into active use were to be given monetary cost my guess is that the "losses" would amount to a lot of money. In a proprietary setting there is also much waisted effort, but here it cost money which is an incentive for management to avoid waisted effort. In a FLOSS setting efforts cost time which is an incentive for the developer to avoid waisted efforts.

FLOSS do not have the marketing power of large corporations[1] and FLOSS do not have the same distribution network as conventional merchandises have. The most important distribution channel for FLOSS is the Internet, and the most important marketing channels is the Internet and the users. Linux distributions like RedHat and SuSe bundles Linux and a lot of other FLOSS programs and sells it through conventional distribution channels, but they are the exception and you are not likely to find them in a software shop in Africa. To effective use FLOSS a descent Internet connection is almost[2] required. There is also a lack of support from the hardware industry. Many hardware producers are reluctant to provide drivers to Linux and even more reluctant to release them under a FLOSS license.

One result of the FLOSS process which is an advantage as the same time as it is a disadvantage is the number of choices available to users. To be

[1] Microsoft have the huge advantage of having their operating system pre-installed on most of the PCs sold.

[2] The Ubuntu Linux distribution ships CDs to anybody on request.

able to chose is generally a good thing, but choices requires some effort to obtain the knowledge needed to make a good decision. It takes time to get a general overview of the FLOSS scene. There is a bewildering number of choices. Take for example the number of Linux distributions. For those who have a passion for programming and computers this can be an acceptable cost. For those who use the computer only to get the job done I suppose it can be more appealing to just have one choice, which is just good enough. One appealing aspect of the one-choice approach which Microsoft offer is the freedom from the burden of having to choose. There is business opportunities in offering this same safety by acting as a proxy for FLOSS software, like Linux distributions are sort of doing, and some consultancy firms are doing for business customers. Still there is a great many Linux distributions and consultancy firms to choose from.

US Patent laws create legal headaches for Linux distribution because they risk being sued if they bundle software which violate a patent. Lawyers earns good money on patent issues because patents is so open to interpretation. In the US you can file software patents (on a algorithm or process), in Europe you cannot do this. Linux distributions generally want to distribute their distribution to the US and have to be sensitive to US patent laws. This is the reason why some software is not bundled on the regular CD, but have to be downloaded separately from the Internet. The patent issue chiefly affect multimedia applications because of patents on compression/decompression algorithms.

5.7 FLOSS in developing countries

In this section I want to look closer into how FLOSS is applied in developing countries. The awareness of FLOSS is rising in developing countries. The reasons for this growing awareness are because of, among other things, the danger of falling into a dependent relationship with software vendors, the potential lower cost of FLOSS, the open standards FLOSS promotes, the ability to inspect code for national security reasons, less vulnerability to viruses and the possibility of fostering home grown computer industries.

5.7.1 Advantages FLOSS offer

FLOSS efforts in developing countries have received attention from the mainstream media, this has created some hype around FLOSS, but a quick web search will show that the hype is not altogether unfounded. Coverage in the media have the tendency to make technology sound like “silver

bullets". FLOSS is not a "silver bullet" for developing countries, it requires hard work, but offer real advantages.

One of the great advantage FLOSS offer is that it removes the control over the software from software development companies. If a government chooses to sponsor a FLOSS project instead of buying it from a company, the government is not dependent on updates from one company. In developing counties in particular, fostering of home grown industries is a major concern. If the government choose to take advantage of the rich body of FLOSS software, and hire people to adapt this software to the needs of the government, you have two advantages. First you build up the capacity of the home grown computer industries, and second you can pay in local currency. As small amount of foreign currency is a problem for many countries, this is an advantage.

It is common for government bureaus to develop their software in-house without releasing the source, but in the longer run it will most likely be a major challenge to maintain the software. If the software on the other hand is released with a FLOSS license more government and non-government organisations can benefit from the software and share the load of maintenance. When the software develops over time by the contribution of many independent actors the software is more likely to stay in sync with changing demands. It may no longer be necessary to rip out old software and exchange it with new software. This is often the case for in-house developed software, because the ones who made the software are long gone, and the code is to difficult to maintain. When the code is developed by many independent actors you have to program in such a way that other can understand what you have done, without too much difficulty.

To use proprietary software legally you will most likely have to buy a license. The license often require companies to pay for every user using the software. Even if the software is placed on a server, and used by clients through a network you have to pay a license fee for every client allowed to connect. With FLOSS you do not have any such license cost. When you have software with a FLOSS license on a server, any number of clients can legally use it. In this way developing countries do not have to pay money from their limited foreign currency pool to a company in the developed world. Piracy in the developing world makes this a weaker argument (Rajani 2003).

Perhaps the greatest advantages with FLOSS, and which indirectly lies behind the other advantages, is the learning potential that lies in FLOSS. Even if the source code is unavailable for people who do not have knowledge in programing, it gives the potential for those who have access to computers and who either can or want to learn programing, an insight into how

programs have been created. Only a selected few can have a direct benefit from this, it is few people who can read code and even fewer that read it. Those few people that read and/or make code, can make the information embedded in the code more available through making tutorials, and teaching good programing practices learned from the code. Using business jargon FLOSS can be seen as a massive donation of intellectual properly rights into the developing world.

If you add open development processes and open documentation to open source, the learning factor becomes even more significant. It is not necessary to have open development process and open documentation in order to call a project FLOSS, but it is often the case that e-mail archives, forums and wikis is open for everybody with Internet access. There exists many tutorials on programming languages and libraries which have been immensely useful to me. If you have a problem chances are that some else have had the same problem and asked about it on a forum/e-mail list, so you can search the www for the answer. If you don't find the answer you can ask the question yourself.

5.7.2 Participation in FLOSS

According to the Hacker Survey only 7,5% of the 519 respondents were from what I identify as a poor country. 42.2% were from Europe and 46% were from the North-America. This show, not surprisingly, that the wealthy and predominately western counties dominates FLOSS. North-America and Europe dominates most sectors of the computer industry, but unlike other sectors FLOSS gives developing countries a chance to catch up. The body of code produced with an FLOSS license is freely available. There are, however, still limiting factors to this freedom. The two most important factors that limits developing countries from contributing to and benefiting from FLOSS is education and Internet access. Those fortunate enough to have education and Internet access in the developing world can most certainly contribute, provided they are sufficiently motivated.

India, China and Brazil are countries know be progressive in their support for FLOSS. In Brazil, FLOSS has strong government backing. Linux is used in a number of *Telecentros* offering public Internet access, among other things. China is know for its Linux distribution, Red Flag Linux. India is known for the Simputer *Personal Digital Assistant* (PDA) based on FLOSS software. All this countries have one thing in common, they are quite advanced developing countries. More advanced countries are better able to effectively participated in the global FLOSS community. Even if poorer

countries can participate less they can benefit from FLOSS as users of the software.

A natural place for countries aspiring to use FLOSS in the developing world to start their participation, is with the localisation of software. The Kiswahili Linux Localization Project (http://www.kilinux.udsm.ac.tz/) is an example of such. This project's mission is to localise FLOSS software. The project has localised OpenOffice and Firefox. As we will later see localisation efforts is also important in Ethiopia.

Ubuntu is an interesting distribution in the African context. Ubuntu was founded by Mark Shuttleworth, a dotcom millionaire from South Africa. Ubuntu focuses on user friendliness and call it self "Linux for human beings". To help people with slow Internet connection to get Ubuntu they send CDs on request, free of charge. The Ubuntu distribution describes it self in this way on its website:

> 'Ubuntu' is an ancient African word, meaning 'humanity to others'. Ubuntu also means 'I am what I am because of who we all are'. The Ubuntu Linux distribution brings the spirit of Ubuntu to the software world.

The Ubuntu project is led by a company named Canonical which is registered on the Isle of Man, so Ubuntu is not strictly African though it has African ties. A more strictly African Linux distribution is Impi Linux which is an Ubuntu gold partner in South Africa. Impi Linux is a commercial distribution including third party proprietary software, unlike Ubuntu.

Rajani (2003) gives an overview of FLOSS efforts in Africa, Latin America and Asia. It is about three year old and do not show the latest developments, but it is still a valuable reference. FLOSS in developing countries is a fast changing field, like FLOSS generally is.

5.7.3 Projects for the developing world

Digital divide arguments and solidarity with the poor in developing countries have inspired the development of low cost computer hardware solutions. In this section I am going to describe and discuss three project aimed at giving computer and Internet access to people who are not able to buy a conventional PC. All of this projects uses FLOSS software as part of their solutions.

The Simputer project (http://www.simputer.org/) has developed a PDA designed to provide affordable computing to poor and illiterate people in developing countries. The Simputer is the first computer designed in India, and was primarily designed with the rural poor and illiterate people of India in mind. Simputer uses a Linux based operating system, and initially had a few applications thought to be relevant in a rural Indian context. A prototype of Simputer was finished in 2001 and since that time two companies, PicoPeta and Encore Software have started to manufacture and sell different version of Simputer. Simputer has not become the success it was hoped to be, initial sales have been low. In response to marked realities the Simputer has been refined to include applications common to regular PDAs. According to a SciDev.net article[3] 31st of July 2006, one of the initial Simputer designers, Swami Manohar, is reported to have said:

> The under-privileged want to have the same technologies the privileged have, not some cheap stuff that do-gooders provide for them.

The story is far from over for Simputer. Simputer was probably the first serious attempt at making an affordable computer. Simputer shows how it is possible to make computers based on FLOSS available to a bigger audience in the developing world. Simputer is based in India, which is a developing country. India is also known as a popular country for outsourcing software development. This shows that a combination of computer knowledge and openly available software and standards can give business opportunities for developing countries. Without FLOSS I cannot see that a project like Simputer would be possible. If all code were closed Simputer would have had to license proprietary software giving less freedom in design and increasing the overall price, or make the operating system and all the applications from scratch which would require a lot of man hours and finances.

The new hot candidate to provide low cost computer access is the *One Laptop Per Child* (OLPC) project (http://www.laptop.org/). OLPC is a non-profit association which was formed by MIT Media Labs and funded by a number of sponsor organisations including AMD, Red Hat and Google. OLPC have partnered with Red Hat to provide software. This project aim at providing laptops to children, the chairman of OLPC, Nicolas Negroponte, says that OLPC is about learning and exploration, not giving kids costly tools and toys.

> OLPC is based on constructivist theories of learning pioneered

[3] http://www.scidev.net/content/features/eng/pcs-for-the-poor-as-good-as-their-hype.cfm

by Seymour Papert and later Alan Kay, as well as the principles expressed in Nicholas Negroponte's book 'Being Digital'[4].

OLPC and associates are in the process of developing a laptop specially designed for children in the developing world. The design goals of the laptop is to make it robust and durable, able to withstand dust and heath, and able to operate where there is little or no electricity. It is going to be equipped with long range wireless hardware able to do ad-hoc *Peer-to-Peer* (P2P) networking, or mesh networking as it is also called. In this way information can be sent between the laptops and if one laptop is connected to the Internet, all the laptops in the mesh get connected to the Internet. It is going to have a low cost dual mode display, one color laptop mode and one monochrome mode for reading books. For electricity it is going to use batteries and perhaps some sort of manual electricity generator, like a hand-crank, in addition to regular power. It is planned to run on a Linux based operating system provided by Red Hat. The price of the laptop is planned to eventually get lower than $100.

A prototype was presented on the *World Summit on the Information Society* (WSIS) in November 2005. Since that time many different prototypes have been made. The OLPC projects relies on large orders from governments to sell this laptop. At this time OLPC only accept order of over 1 million units, and production will not start before the orders have passed 5 million units. Several countries have shown interest; Nigeria, Brazil, China, India, Egypt, Argentina and Thailand are among them. India has pulled out of the project. The Indian Ministry of Education dismissed the laptop as "pedagogically suspect"[5].

This ambitious project has received a lot of press, being praised and criticised. The design of the laptop is innovative, and the price tag is appealing compared to existing alternatives. It is an interesting device which can, among other things, make more books available to children and their family, if books are allowed to be distributed electronically for little or no charge. A major flaw with OLPC is that the design and distribution is done in a top-down manner. The project with its constructivist pedagogy and sole focus on children, can easily be perceived as a way to impose US liberal values on the developing world. There seem to be an overall lack of cultural and contextual sensibility in this project. Lee Felsenstein at the Fonly Institute identifies some thought through problems with the OLPC approach[6].

[4] http://wiki.laptop.org/go/One_Laptop_per_Child
[5] http://www.theregister.co.uk/2006/07/26/india_says_no_to_olpc/
[6] http://fonly.typepad.com/fonlyblog/2005/11/problems_with_t.html

Negroponte is known for his techno-utopianism, apparent in his book "Being Digital". Negroponte has a techno deterministic view of technology, the laptop is imagined to have a predetermined effect, where for example the children learn about mathematics by making a program that draws a circle. Resistance to the laptop is perceived as backwardness and technophobia, children do not have all this "baggage" and are more receptive to technology. This is one of the reasons the project only focuses on children. Experiences obtained from IS research show that the effect of technology is not predetermined, but dependent on the context within which it is applied. Some of the children will perhaps only use the laptop to play games, and as a consequence homework and house duties will not be done.

The sole focus on children without regard to the family, teacher and local community can also become a problem. The laptops can empower children at the expense of the family and the teacher. The teacher should at the very least have a laptop too. There are more pressing needs in many schools in developing countries, and the value of laptops in teaching and learning is uncertain. I guess this and the constructivist pedagogy, are some of the reasons the Indian Ministry of Education calls this project "pedagogically suspect". Even if the laptop manage to sell in sufficient quantity, it still needs software useful for children in various countries, software that facilitates learning not time-killing. Despite all this reservations it is clear that OLPC has at the very least been successful as a marketing campaign by raising attention to low-cost computing for developing countries.

The last project I am going to review is Ndiyo (http://www.ndiyo.org/). Ndiyo is the Swahili word for "yes". Ndiyo is a non-profit organisation based in Cambridge UK. The vision of Ndiyo is to provide low-cost networked computing, making computing affordable to more people. The fundamental idea to provide this is through ultra-thin-client computing and FLOSS. Thin-client networking is a solution with one powerful server, having all or most of the software, and a number of less powerful clients with minimal software. In the ultra-thin-client scheme promoted by Ndiyo the thin client do not have any hard-disk and a minimal set of I/O devices. Nidyo started with a monitor, and asked what was the minimum they could add to make it into a workstation. A company called Newnham Research was created based on this idea, and they invented the *nivo* (Network In, Video Out). This is a box to which you can connect a network cable, a monitor, a mouse and a keyboard. This device get compressed pixels from the server which is decompresses and sent to the monitor, so all computing is done on the server. Using the nivo boxes and Ubuntu Linux on the server, Ndiyo has come up with a working solution.

The ultra-thin-client solution is similar to the terminals of the mainframe

area, but with thin-clients the user can get individualised displays. The PC has now become so powerful that a single one is powerful enough for many users. That is if the users do not play graphics intensive games, do video editing or make complex 3D animations. The ultra-thin-client computing solution is cheaper than the conventional networked PCs, but the greatest advantage in my opinion is the lower maintenance burden. There is nothing in this solution that mandates the use of FLOSS, but the use of FLOSS lower the total price of the thin-client network. This seems like a reasonable solution for Internet cafes, tele-centers, small offices and computer rooms in schools. Newnham Research currently market something they call the USB nivo, which is a device to connect monitors to a PC using an USB 2.0 port. Newnham Research seems to have distanced themselves from Ndiyo, Ndiyo is not mentioned once on their current web site.

All three of this projects are idealistically motivated to provide computing to poor people through low-cost hardware. Even if this projects are idealistically motivated there need not be anything idealistic about providing low-cost hardware. Providing low-cost hardware can be seen as a way to leverage emerging markets, low-cost hardware sold in large quantities can generate a handsome profit. The big hardware vendor Intel and the big software vendor Microsoft, have got their eyes opened to a potential marked in lower cost computing. Intel has started the *World Ahead Program* and is closely collaborating with Microsoft to provide lower-cost computing. Intel has made a laptop marketed as a learning device, which was formerly codenamed Eduwise. Microsoft has a PC purchasing model called FlexGo on the drawing board. With FlexGo the customer only pay a part of the cost for a new PC up-front, and the rest is paid by prepaid cards or a subscription with monthly payments. Intel has promised to invest $1 billion in India during the next five years. The World Ahead Program is a key part of this investments.

5.7.4 FLOSS participation and use challenges

In this section I will try to identify some of the challenges that are unique to developing countries when it comes to both using FLOSS software and participating in the FLOSS community.

Education is known to be an important factor for developing countries to be able to grow. This becomes even more true as we move into a post industrial era where ideas and manipulations of ideas becomes an increasingly important part of the global economy. FLOSS is at its very core a prime example of how ideas and manipulation of ideas can be used to create value. To be able to benefit from the opportunities that FLOSS provide, a certain

level of knowledge is demanded. The ability to use software applications certainly requires some knowledge from the user, and perhaps even more to be able to use certain FLOSS applications. However, to be able to not only use, but to contribute to FLOSS and localise FLOSS applications, a much higher level of knowledge is required.

Considering, as the case is in Ethiopia, that only towns and cities have electricity there is for most people in the developing world a long way to go before computers becomes a commodity. Even if I am a computer scientist who like computers, I have to admit that for many people there are more pressing needs than getting a computer. Since the number of computer users with an Internet connection compared to the total population is lower in developing countries, there is also less people to be attracted into use of FLOSS and participate in FLOSS.

On a practical level I can mention one thing that can limit the adoption of Linux in the developing world; the lack of support for many softmodems. Softmodems are low-cost modems where most the of work is done in software. It is difficult to make drivers for this devices because so much has to be done by the driver. Many softmodem producers are reluctant to release drivers to Linux. In the developed world broadband access to the Internet has become so common that there are little interest among FLOSS developers in making softmodem drivers, there basically is no "itch" to "scratch". In a weblog called Meskel Square, Andrew Heavens faced this problem when installing Ubuntu Linux on a laptop[7]. Dial-up access is still the most common and sometimes the only mean, to connect to the Internet in developing countries. Internet access is a necessity to participate in FLOSS, and a great advantage to effectively use FLOSS.

There are also broader social and political issues that can limit the applicability of FLOSS. (Rajani 2003) mentions bureaucracy, corruption, "brain drain", political freedoms and legal framework. Many of this issues do not directly limit FLOSS, but can have an indirect influence. Rajani mentions bureaucracy as perhaps the most fundamental barrier to wider FLOSS adoption. It is common to think of bureaucracy as constraining, but it can be enabling as well. Bureaucracies can be fundamental in implementing plans for wider adoption of FLOSS.

[7] http://www.meskelsquare.com/archives/2006/06/ubuntu_the_diff.html

Chapter 6

Health Information Systems Programme (HISP)

The Health Information Systems Programme started as a pilot project in South Africa in the wake of the apartheid regime. The legacy of apartheid was inscribed into the health information systems of South Africa. In its first phase HISP was a collaborative research project between research and health institutions in South Africa and Norway. Over the years HISP has evolved into a global *Health Information System* (HIS) network with nodes in South Africa, Norway, Mozambique, Ethiopia, Vietnam, Tanzania and India.

HISP want to promote the usage of empirical founded information for decision making in the health sector. The overall goal is to empower the poor and the marginalised of the world. Based on experience from South Africa and other nodes in the network, guidelines on the kind of information that should be gathered and best practices in gathering and applying this information has been made. On a more practical level the software DHIS has been developed to support data gathering and analysis.

In this chapter we will first step through important steps in the history of HISP. Then I will present the philosophy forming and the methods promoted by HISP. This methods together with DHIS is being exported by HISP to various countries willing to give it a try. I will give a short introduction of the design goals behind DHIS, and how it helps to promote the overall goals of HISP.

6.1 HISP history

The first truly democratic election in South Africa, with no discrimination on race, was held in 1994. *African National Congress* (ANC) won the election and launched the *Reconstruction and Development Program* (RDP). This program was initiated as a broad program which involved many different parts of the South African society. The overall goal for the program was to remove the relics of the former apartheid regime and build a free and non-discriminating society.

The health system built during the apartheid regime was clearly discriminating on race. There were fourteen different departments of health at the central level. The "general" department, and "white", "Asian" and "coloured" departments, and ten for "blacks", "homelands" and "self-governing states". To reform the health system several projects were initiated. HISP, which focuses on health information, was initiated to this end.

HISP was established as a collaborative research and development effort between University of the Western Cape, *University of Oslo* (UiO), and University of Cape Town. In 1996 HISP received founding from *Norwegian Agency for Development Cooperation* (NORAD) for a pilot project in three health districts in the Cape Town area. Two areas for research and development were identified. The first was to develop an *Essential Data Set* (EDS) and standards for primary health care data, the second was to develop what would become DHIS. The relative success of HISP compared to two other health information projects in South Africa lead to the official endorsement of HISP. In 1999 to 2001 the software DHIS along with the processes promoted by HISP were rolled out to primary health units in the whole of South Africa.

Since the time of the pilot phase in Cape Town countries outside South Africa have involved themselves in the HISP network. Mozambique have been involved since 1998, India since 2000, Malawi since 2000, Tanzania since 2001, Cuba since 2002, Mongolia since 2002, Ethiopia since 2002 and Vietnam since 2004. HISP is not Cuba and Mongolia at this time. HISP also has some relations with Nigeria and Botswana.

The South African version of DHIS was taken to different countries and adapted by PhD and master students and to a lesser extent ICT professionals, to the local requirement of the country. The way of entry into the different counties has been through two major entry points. The first through university collaboration, and attempting to build alliances with the health authorities. The second entry has been though the departments of health, typically with support from outside actors like donor agencies

(Braa et al. 2004). The entry to Ethiopia has been through collaboration with *Addis Ababa University* (AAU).

6.2 HISP philosophy, methods and processes

The word describing the overall goal for HISP and the reason for HISP's existence is *empowerment*. HISP want to empower the poor and marginalised through strengthening local health services in the developing world. Through training district health staff in gathering and analysing data, both for local use and reporting, HISP hopes to empower the district staff to give better service to their health district.

HISP is founded within the academic community and is therefor both research and action oriented. The research taking place within the HISP network is almost exclusively founded within the discipline of action research. The PhD and master students involved in HISP are adapting DHIS for the local context, train users, negotiate with health authorities and health workers and similar tasks needed to transfer the HISP methods and DHIS to a new region or new country. At the same time the researchers are collecting data for their thesis. There are health workers, ICT professionals and managers within the HISP network who is not tied to the academic community, but HISP have strong ties to this community.

Important to action research within the field of IS research is to involve all the people affected by the IS into the design, development and implementation of the system. The intension behind involving the affected parties into the process is to build a system more attuned to the users needs, and to instill a feeling of ownership towards the IS, thereby increasing the acceptance of the system. This is perceived more "democratic" than building a system in splendid isolation, and more likely to meet user's needs. In the IS literature this is called *participatory design*.

An important factor for HISP in South Africa was the development of an EDS. Often the collection of data in the health systems in developing countries are data centric, which by consequence leads to lot of data being collected with little thought on the usefulness of the data. HISP promotes an action oriented approach to selecting which data to collect. Only data which can give information leading to action should be collected. This have given more focus to indicators and the data needed to calculate them. One example is immunisation coverage. To calculate this indicator you need to know how many children under one year have been fully immunized and how many children under one year living in the area. By challenging the data collected on usefulness an EDS is developed.

To empower local district staff and to lessen tension between what is perceived useful information by different actors in the health system, the model seen in figure 6.1 is promoted by HISP. The collection of routine data is often used to monitor and control the performance at lower levels in the hierarchy. This is an important part, but HISP want to promote local use of information. The combination of data used to report to higher levels and data perceived relevant only at the local level gives the hierarchy of standards for EDS, indicators and procedures in the figure. The size of the EDS grows less as it travels up the hierarchy. To support local use of information it is important that information is travelling both ways in the hierarchy. A region gathering data from the districts needs to report back data on the other districts to each respective district. In this way a district can compare itself to other districts, the regional total etc.

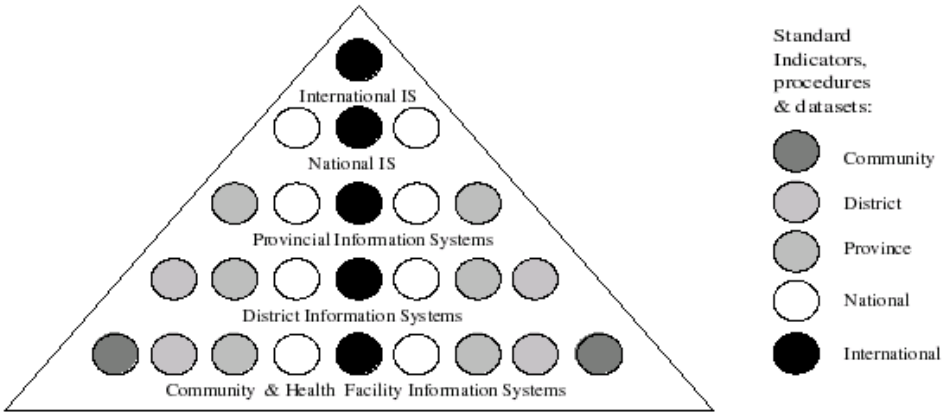

Figure 6.1: HISP hierarchy of standards

A multiplicity of simultaneous processes are taking place within HISP at the same time. There are nearly never any clear start or end to the processes taking place. The process of transfer takes different forms in the countries HISP are transfered to. The transfer of HISP to a new country is called a horizontal transfer. Within a country the process of vertical transfer are taking place. When HISP is transfered to a new region or districts, new and distinct processes are taking place. In Ethiopia the transfer of HISP to Addis Ababa was different from my experiences transferring HISP to the Tigray region. Three broad categories of processes taking place can be identified, however.

1. **Garnering political and financial support** First access to initiate HISP in a new country must be given. This often happens through university agreements or through the department of health. Within the

country the HISP network expands through inter-institutional linkage, and through people and institutions championing HISP.

2. **HIS design, development and adaptations** To adapt DHIS to a new country, standardisation through developing an EDS and indicators is needed. DHIS then need to be adapted through filling in the organisational hierarchy of the health system, the data elements, the indicators, and other adaptations demanded. This is done as a participatory process.

3. **Education and training** To be sustainable, knowledge about HIS and skills with HIS need to be developed and disseminated. This happens through developing educational schemes to support national HIS development, building master's programs in informatics and public health, and through support from the PhD and master programs at UiO and other HISP countries.

(Braa et al. 2004)

A paper authored by prominent actors within HISP (Braa et al. 2004) calls the approach used by HISP the *networks of action* approach. They conclude that this approach is better at addressing sustainability of intervention and is characterised by:

- Abandoning singular, one-site action research efforts in favour of a network of sites.
- Generating local, self-sufficient learning processes and distributing appropriately formated experiences between sites.
- Nurturing a robust, heterogeneous collection of actors likely to pursue distinct, yet sufficiently similar agendas.
- Aligning interventions with the surrounding configuration of existing institutions, competing projects, and efforts as well as everyday practises.

6.3 Inscription of the HISP approach into DHIS

In this section I seek to describe how the philosophies, methods and processes described in the previous section has been inscribed into the DHIS software. DHIS as an artifact is very important to the spread and practical applicability of the HISP approach. This analysis is based based on the MS Access line of DHIS, but I will try to make it sufficiently abstract to avoid

getting into the technical particularities of the different versions of DHIS. That I will come back to in the empirical part of this thesis.

Important to the spread of HISP is vertical scaling, relating to the hierarchy of standard seen in figure 6.1. To make DHIS scale to all levels within a county's health system an organisational hierarchy are inscribed into the system. Data can be fragmented down to the level of the reporting unit, and data can be aggregated for all levels above the reporting unit. To facilitate local use of information, and at the same time facilitate reporting to the level above, a unit in the hierarchy can have any numbers of data elements and indicators. Reporting is done by exporting data for data elements and indicators required by the level above. This can then be imported into a DHIS database at the level above.

To empower local health worker through local use of information, DHIS offer support for data analysis at all levels. DHIS offer support for calculating indicators and making charts. Raw data, indicators and charts can be shown at all levels of aggregation from the reporting unit and up. A district information officer having received data from all the other districts in the region can compare data from his district to that of the regional total and with other districts. As mentioned data needs to travel both up and down in the hierarchy to make this possible. DHIS also offer support for validating data to avoid mistakes and inaccuracies in reporting. This is needed to make the data more reliable.

To facilitate horizontal scalability, that is transfer of DHIS to new countries, DHIS is designed to be flexible. DHIS is not a ready made system, but needs to be configured to be useful in a new country. The health hierarchy and all the health institutions have to be specified. All the indicators, semi-permanent data (non-routine data used for indicator calculation etc.), data elements, reports etc. have to be specified. DHIS is designed to make building a database tool for a new country easy. To facilitate translation of the user interface into new languages strings are not hard coded into the system, but retrieved from a database containing the strings in different languages. DHIS is basically divided between two parts; a stable core and a flexible exterior. The stable core is the data structures and processes being stable across different health systems. An hierarchical organisation with clinics, hospitals etc., and administrative units are found in all modern health systems. The core parts of DHIS are the health system hierarchy, the general concepts of data elements and indicators and the processes of reporting and data analysis.

The MS Access line of DHIS (DHIS 1.x) is made with tools available in MS Office. It uses MS Access, *Visual Basic for Applications* (VBA) and MS Excel. The reason for choosing MS Office as the platform for DHIS was based

on pragmatic considerations. The development of DHIS was initiated as a part of the first pilot phase of HISP in South Africa in 1996 - 1998. MS Office was already in widespread use in South Africa, with or without a license, so the cost of obtaining MS Office was not considered a problem. MS Office is an expensive office suite, but software piracy is so widespread in many of the countries HISP operates that this cost have not become a serious problem yet. Because MS Access is a RAD tool, it is a good tool for rapid prototyping. The ability of HISP to get a working database tool made within a relative short time was an important factor in making HISP a relative success to the other HIS initiatives in South Africa. DHIS together with *pivot tables* in MS Excel made DHIS more than a database tool. Pivot tables offers a powerful tool for data analysis, giving user the ability to make charts and diagrams of the data. The ability of MS Access to integrate with other MS Office programs, like Excel, made the use of MS Access seem like a natural choice. The availability of programmers skilled in VBA also came into the consideration (Braa and Blobel 2003).

Part IV

Empirical Study

Chapter 7

The Ethiopian Context

In this chapter I will give a presentation of Ethiopia. This presentation will be geared towards ICT and information relevant to FLOSS and HISP. Ethiopia has a long history with many kingdoms having come and gone in the past two thousand years. Ethiopia is currently one of the worlds poorest nations with a GDP per capita at $800, and have been plagued by war, drought and civil unrest for decades.

7.1 Demographics

Ethiopia	
Capital	Addis Ababa
Official Language	Amharic
Government	Federal republic
President	Girma Wolde-Giorgis
Prime Minister	Meles Zenawi
Population 2005 est.	73 mill
Area	1 127 127 km^2
GDP 2005 est.	
Total	$60 billion
Per capita (PPP)	$800
Life expectancy 2006 est.	
Male	47.86 years
Female	50.24 years
Literacy 2003 est.	
Male	50.3%
Female	35.1%

The Federal Democratic Republic of Ethiopia is the largest country on the Horn of Africa. Ethiopia is a mountainous country with a high central plateau cut in half by the Great Rift Valley. A map of Ethiopia can be seen in figure 7.1. Ethiopia is an ethnically diverse country with more than 80 different ethnic groups speaking 84 indigenous languages. It is also one of the worlds poorest nations, with around half the population living under the poverty line. The infrastructure of Ethiopia is poorly developed.

Ethiopia is a country with long history and was the home of the Axumite kingdom. The Axumite kingdom was technically advanced for its time. Ethiopia has developed its own script, the Ge'ez script, and its own written languages. The name Ge'ez come from an ancient language now only used in the liturgy of the Ethiopian Orthodox church. Ge'ez also refers to the script used to write Amharic and other languages in Ethiopia. In Ethiopia this script is called *Fidel*.

Ethiopia uses its own calendar system. The Ethiopian calendar is similar to the Julian calendar used in eastern orthodox churches, but with the significant difference that it has 13 months. The first 12 month have 30 days and the last month have 5 days in regular years and 6 in leap years. Thus the length of the year is the same as in the Julian calendar, which have 12 months of irregular sizes (30, 31 or (28/29)). The Oromo people have yet another calendar.

All the population related data are uncertain. A census have not been made since 1994. The current size of the population and the percentage of different ethnic groups and religion are all estimates. The percentage of the different ethnic groups and religions are politically sensitive and therefore it is reason to be critical to the estimates.

The Oromo and the Amhara people are the two domination ethnic groups in Ethiopia and the Tigrayan people are mostly living in Tigray, where I worked. According to the CIA World Fact Book 40% belonging to the Oromo people, 32% belonging to the Amhara and Tigrayan people. According to the 1994 census 32.1% identified themselves as Oromo, 30.2% identified themselves as Amhara and 6.2% identified themselves as Tigrayan. I have also found a survey from 2005 which focused on demographics and health ((Director) 2005). This survey was made on a representative sample of 14 645 household all over Ethiopia. 14070 women and 6033 men responded to this survey. From the respondents to this survey 32.6% belonged to the Oromo people, 31.3% belonged to the Amhara people and 6.8% belonged to the Tigrayan people.

When it comes to the religious affiliation the difference between the World Fact Book on one side and the 1994 census and the demographics and

health survey on the other is even bigger. The CIA World fact book says there are 45%-50% Muslims, 35%-40% Ethiopian Orthodox, 12% animists (traditional) and 3%-8% others. The 1994 survey says there are a total of 61.6% Christians (50.6% Orthodox, 10.1% Protestant and 0.9% Catholic), 32.8% Muslims and 5.6% traditional. The survey says a total of 68.7% Christians (49.2% Orthodox, 18.3% Protestant and 1.1% Catholic), 28.8% Muslims and 2.4% others.

My point in listing this variations of the presumed percentage of ethnic groups and religious affiliation is to show how uncertain data from a poor country like Ethiopia is. It seams, however, that the CIA World Fact Book is way of in its estimates. I find this a little disturbing because I thought the Fact Book to be more accurate. The CIA World Fact Book have frequently been cited in papers I have read.

The infrastructure is poorly developed only 13.4% of the population have access to electricity and only 6.1% of the households are electricity customers. Tigray region have better electricity supply than most of Ethiopia. The road density in Tigray is in the vicinity of the national average. Tigray is a mountainous region where it is expensive to build roads.

The official language of the federal state is Amharic. The official language of the regions is decided by the respective region. The language of instruction in the primary school (grade 1 to 8) are decided by the region. After grate eight the language of instruction in schools is English. The enrolment rate is very low with little over half of the children in school age being enrolled.

The economy of Ethiopia is based on agriculture, which accounts for half of the GDP, 90% of exports and 80% of total employment. Coffee is the major export crop providing for 65%-75% of Ethiopia's foreign exchange earnings. Ethiopia experience periodic droughts. In 1984-85 there were severe famine in Ethiopia caused by drought and political instability. Currently there are drought in the Horn of Africa, but this have not lead to a famine in Ethiopia yet. The rainy season is from mid-June until mid-September. Ethiopia was ranked by the UN to be the 170th of 177 countries measured by *Human Development Index* (HDI) in 2005. HDI is a measurement aspiring to measure the quality of life according to health, knowledge and wealth.

7.2 Ethiopia, a Land of History

Current day Ethiopia has a long historical heritage. The Ethiopian civilisation has had many different names and expanded areas in modern day

Sudan, Yemen, Djibouti, Eritrea, Ethiopia and Somalia. The first records of Ethiopia stems from Egyptian traders and dates from 3000BC. The state of Sheba is believed by some to has been located in Ethiopia, but the location is disputed between Yemen, Ethiopia or both. From the 19th century up until after World War II Ethiopia was called Abyssinia. The information for this historical summary I have got from the English Wikipedia.

The first verifiable kingdom of great power to rise in Ethiopia was the kingdom of Axum. Axum grew from 5th century BC to become a important trading nation by 1st century AD. The capital was in the city of Axum. Christianity was introduced into the country in 330AD by Frumentius. The Greek boy Frumentius was taken captive and sent to the King of Axum as slave where he gained favor with the court. The Axumite kingdom lasted until 11th or 12th century when it was succeeded by the Zagwe dynasty.

Yekuno Amlak overthrew the last Zagwe king in 1270 and introduced the Solomonid dynasty. He claimed ancestry from the last king of Axum. The Solomonid dynasty claims decent from king Solomon and the Queen of Sheba, who is said to have given birth to the traditional first king Menelik I after her Biblically-described visit to Solomon in Jerusalem. This dynasty lasted until 1974 when the last emperor, Haile Selassie, was deposed.

Towards the close of the 15th century the Portuguese missions into Ethiopia began. A belief had long prevailed in Europe of the existence of a Christian kingdom in the far east, whose monarch was known as Prester John, and various expeditions had been sent in quest of it. Among the explorers who had engaged in this search was Pedro de Covilham, who arrived in Ethiopia in 1490, and, believing that he had at length reached the far-famed kingdom, presented to the emperor of the country, a letter from his master the king of Portugal, addressed to Prester John. This relation would prove helpful when the Somali General and Imam, Ahmad ibn Ibrihim al-Ghazi, attacked Ethiopia from 1528 until he was defeated in 1543. Portugal sent 400 musketeers to aid the Ethiopian emperor.

Ethiopia remained independent during the *Scramble for Africa* and are the only country in Africa which has never been colonised. Ethiopia was, however, not unaffected by the *Scramble for Africa*. Menelik II, emperor of Ethiopia from 1889 to 1913, exchanged the region Eritrea in 1889 for rifles, ammunition and cannons with Italy. This agreement was disputed. The Italian version of the treaty said that Ethiopia had become an Italian protectorate, but the Amharic version said that Ethiopia had become an extended partner with Italy under Menelik II's full authority. This conflict came to head with the *Battle of Adowa* in 1896 where the Italians were defeated.

In 1930 the last emperor in the Solomonid dynasty Haile Selassie came to

power. His reign was interrupted between 1936 and 1941 when Ethiopia was under Italian occupation. When Selassie returned to power he started on a program of modernisation. He improved diplomatic ties with the US, and sought to improve the nations' relationship with other African nations. Selassie's rule ended when he was deposed in 1974 after a period civil unrest. A provisional administrative council of soldiers, known as the Derg ("committee") seized power.

Mengistu Haile Mariam assumed power as head of state and Derg chairman, after having his two predecessors killed. Mengistu's years in office were marked by a totalitarian-style government and the country's massive militarization, financed by the Soviet Union and the Eastern Bloc, and assisted by Cuba. The Derg proclaimed itself to be socialist, and gradually it turned Ethiopia into a communist country. In 1984 it formed the *Workers' Party of Ethiopia* (WPE). In 1987 a new Soviet-style civilian constitution was introduced and Mengistu became president.

During the reign of Haile Selassie sentiments had stirred in Tigray. Many of the Tigre people felt that they were being treated unfairly by the central government. This eventually lead to the formation of *Tigrayan Peoples' Liberation Front* (TPLF) in the first half of the 1970s. TPLF continued the struggle against the central government during the Mengistu regime. TPLF had ties of cooperation with *Eritrean People's Liberation Front* (EPLF) dating from even before the formation of TPLF. EPLF gave military training to TPLF during the time of TPLF's formation. In the 1980s, TPLF had a reputation as hard-line communists who saw Enver Hoxha's Albania as a model state. Observers used to joke that when TPLF took control over a town it would take down the portraits of Marx, Engels and Lenin in government offices, and replaced them with even larger ones. In 1989, the Tigrayan Peoples' Liberation Front (TPLF) merged with other ethnically-based opposition movements to form the *Ethiopian Peoples' Revolutionary Democratic Front* (EPRDF). In this process TPLF abandoned a secessionist agenda and the former hard core Stalinist ideology (Berhe 2004). In May 1991, EPRDF forces advanced on Addis Ababa. Mengistu fled the country and was granted asylum in Zimbabwe, where he still resides.

A transitional government was formed by EPRDF and Meles Zenawi, from TPLF, was appointed President. Most opposition groups boycotted the election held in 1995, ensuring EPRDF a land slide victory and giving Meles Zenawi the position of Prime Minister. EPRDF retained majority in 2005 general election, but lost many seats to the opposition. Meles Zenawi is still in office.

TPLF's former allies EPLF took control of the province of Eritrea, the northernmost part of Ethiopia, and formed the state of Eritrea in 1993. Even if

this made Ethiopia a land locked nation, the independence of Eritrea was recognised by the transitional government of Ethiopia. The border was, however, un-demarcated. In 1998 border disputed lead to the *Eritrean-Ethiopian War* which ended in 2000. The relationship between Ethiopia and Eritrea is still tense and the border dispute is still not settled.

7.3 Politics

Ethiopia is governed as a federal parliamentary republic. The federation consist of 9 semi-autonomous regions, divided according to ethnicity, and two city regions. The regions have power to raise and spend their own revenues. The parliament have two chambers; *House of People's Representatives* (the lower chamber) with 547 members and *House of the Federation* (the upper chamber) with 110 member. The presidency acts as head of state and is elected by the House of People's Representatives and House of the Federation for a six-year term, but have little political power. The prime minister is chosen by the majority group in the House of People's Representatives. The Council of Ministers is selected by the prime minister and approved by the House of People's Representatives.

The political system in Ethiopia is divided into an executive, legislative and judicial branch. The executive branch is the president, prime minister and the council of ministers. Most political power is held by the prime minister. The legislative branch is the parliament. The judicial branch is more or less independent. The president and vice president of the Federal Supreme Court are recommended by the prime minister and appointed by the House of People's Representatives; for other federal judges, the prime minister submits candidates selected by the Federal Judicial Administrative Council to the House of People's Representatives for appointment.

An important and disputed characteristic of the current political system in Ethiopia is the federal system, with regions drawn according to ethnic lines. The constitution even give the right of secession to any nationality within Ethiopia. This is not surprising considering that TPLF started out as a secessionist movement and joined forces with other ethnically based movements. At blog sites such as blogspot.com many posters express concern about ethnic division in Ethiopia and some accuses Meles Zenawi for using divide and conquer strategies.

The regions in Ethiopia are semi-autonomous. Each region have a Regional State Council which is elected by popular vote. The election for the House of People's Representatives in the federal parliament is held every five

years. The election of the regional councils are conducted by the respective region. The members of the House of the Federation are elected by the regional councils, and is chosen for a 5 year term. The current regions in Ethiopia are the following including two charter cities:

- Addis Ababa (charter city)
- Afar
- Amhara
- Benishangul-Gumaz
- Dire Dawa (charter city)
- Gambela
- Harari
- Oromia
- Somali
- Southern Nations, Nationalities, and Peoples Region
- Tigray

In the 2005 election EPRDF got the majority votes and holds 327 of the 547 seats in the House of People's Representatives. The two major opposition groups *Coalition for Unity and Democracy* (CUD) and *United Ethiopian Democratic Forces* (UEDF) got 109 seats and 52 seats respectively. Allegation of election fraud were raised by the opposition. Unrest stirred up in Addis Ababa in protest against the alleged fraud. The government cracked down hard on the protesters, leading to close to hundred deaths and several hundreds arrests.

7.4 ICT in Ethiopia

In this section I will explore the current state of ICT in Ethiopia, and the plans and aspiration the government have for the future. The Ethiopian government aspire to leapfrog Ethiopia into the information age. Ethiopia is too poor not to make use of ICT it is said. There is a noticeable aspiration in the African continent to leapfrog the continent into the information age. This can be seen by the quite rapid building of mobile phone networks, bypassing the fixed line telephone technology.

It is true that the current telecommunication infrastructure is poorly developed. Few have a telephone in their house, in 2003 there was only 435,000 main telephone lines in Ethiopia according to the World Fact Book. The number of mobile phone subscribers was only 178,000 in 2004. Report has it that the number of mobile users are increasing rapidly. The state owned *Ethiopian Telecommunications Corporation* (ETC) are increasing the capacity of the mobile network, and according to The Reporter the number of mobile subscribers has increased to 510,000. This is still few even by African standards, however. To rectify this the current government in Ethiopia have grand visions.

In a presentation about e-readiness (Docktor) a number of African countries are measured according different indicators for e-readiness. This presentation uses a five point scale for measuring the level of e-readiness for each of the indicators: Low, Low-Medium, Medium, Medium-High and High. Here I present some of the results for Ethiopia.

PC penetration	Low
Bandwidth	Low-Medium
Government web pages	Medium
Secured servers	Low
Gross enrollment ration in education	Low
IT students in tertiary education	Medium-High
High tech exports	Low

According to an article at Guardian Unlimited, Ethiopia is using 10% of it's GDP every year on ICT. The government plans to invest more than $100 million on pubic sector computing in the next five years. The article is based on an interview with, among others, Ethiopia's prime minister Meles Zenawi. In the article Meles is cited to say (Cross 2005):

> I want to see ICT pervade all our activities as a government, not just in the urban areas. We want to connect all our villages in two to three years. All education services, likewise. We would also like to provide a bit of tele-medicine.

4,000km of optical fiber lines has already been laid out, this lines connects all the regional states in Ethiopia. It is planned that by 2007 none in Ethiopia will live more than a few kilometers from a broadband access point. This network are going to support two major ICT initiatives, Schoolnet (http://www.schoolnet.et/) and WeredaNet.

Schoolnet is an education network which would provide more than 450 secondary educational institutions with access to general ICT, e- mail, and the

WWW. It would allow these institutions to receive streamed Internet- and broadcast TV-based educational content from media agencies. The hope is that this will improve the quality of education and help to overcome Ethiopia shortage of qualified teachers. Some schools already receive video lessons broadcast for eight hours a day by satellite TV.

WeredaNet connects 600 Weredas to 11 regional capitals. This network supports IP telephony and video conferencing, in addition to more traditional Internet traffic like WWW and e-mail. Half of the links is by cable and half by satellite connection. The hope is that this will improve public service. Previously official reports could take months to reach the capital. This network was mobilised to train officials for the general election in May 2005. A web site showing the result of the election is available at http://www.electionsethiopia.org/.

There are also plans to connect 30 research and operational agricultural centers and to connect all major referral hospitals to form the basis of a national telemedicine infrastructure. Through this grand plan for the improvement of Ethiopia's ICT infrastructure the Ethiopian government hopes to leapfrog Ethiopia into the information age. Ethiopia do not have any significant legacy systems to worry about and can apply modern telecommunication, multimedia and data transfer technologies.

To implement this grand vision Ethiopia, ETC and the ministry of Capacity Building have signed contracts with several companies. The previously mentioned 4,000km of fiber lines have been built by Alcatel, *China International Telecommunication Construction Corporation* (CITCC) and Siemens. This lines have been laid to support mobile and fixed telephone users, and for data traffic. The plan is to lay more than 10,000km of fiber optics lines

A more data traffic related project have been implemented by a South-African company called Business Connexion. Business Connexion is a Cisco gold partner. Using Cisco technology a data network with it's core in Addis Ababa have been made. This network consist of a 16-node fiber optic ring around key sites in Addis Ababa. Using combination of telephone carriers, microwave links and satellite links this network is connected with all the Wereda offices in Ethiopia. This network are designed to handle multimedia and data traffic. For data traffic this network will have a carrying capacity of approximately 100,000 Internet customers and provide for 2 000 dedicated lines by *Asymmetrical Digital Subsciber Line* (ADSL), *Fixed Wireless Access* (FWA) or optical fiber with metro Ethernet connection.

To further boost the development of ICT in Ethiopia support have come through a *Technology, Entertainment and Design* (TED) Prize. TED is an annual US event. This prize was awarded to Bono from the music band U2 in

2005. One of the three wishes to be awarded him was to connect every hospital, health clinic and school in Ethiopia to the Internet. *Advanced Micro Devices* (AMD) is one company that as stepped up to fulfill, at least parts, of this wish.

Ethiopia is currently not connected to the Internet backbone, but all international Internet traffic is routed via an US satellite ISP operator. By using a program called `traceroute` I have done a little research on how packets going into Ethiopia is routed. I found that IP packets from Norway to Ethiopia (www.ethionet.et) is routed via Intelsat. Intelsat is an international satellite communication operator. The IP packages are routed to the US and from there to the UK where the package enters Intelsat's network and is sent by satellite to Germany and from Germany to Ethiopia by satellite. The response time from Ethiopia is significant, between 600 and 700ms. I have tried to find the bandwidth of Ethiopia's connection to the international Internet. The most resent data I could find is from 2002. Then the bandwidth was 10Mbps for incoming traffic and 4Mbps for outgoing, for the whole country.

Even if the internal network in Ethiopia have improved significantly over the last few years it would be of little help to Internet connectivity if the connection to the broader Internet is as low as 10Mbps. To give more bandwidth to the Internet a different solution than a satellite connection is needed. Currently there are plans to connect to a fiber optic line going through the Red Sea and Gulf of Aden, this line is called *East African Submarine Cable System* (EASSy). Ethiopia, however, is a land locked nation and have to make deals with it's neighboring countries. Eritrea is not an option given the resent violent history and troublesome dealings with that country. According to the Ethiopian newspaper The Reporter the Ethiopian government have reached and understanding with Djibouti and Sudan, and the plan is to have a connection both through Djibouti and Port Sudan. When this have been realised Ethiopia have truly joined the international virtual community. Such progress, however, are only available to the elite in developing countries. When even using Internet Cafès is too expensive for the majority of the population, not to mention the knowledge level required to use computers, the benefit of Internet is inaccessible to most people.

The New Partnership for Africa's Development (NEPAD)(http://www.nepad.org/) have a project called the E-school project. It was first announced in 2003 and aims at connecting schools across Africa to the NEPAD e-school network and the Internet. According to a new bulletin at nepads web site the project target is to connect all high schools within five years and all schools within ten years. Around 600 000 schools need to be connected to reach this

target. This project have attracted support from heavy software and hardware giants like HP, Microsoft Corporation, satellite operator INMARSAT Limited, Oracle Corporation and Cisco Systems. This corporations are the private sector partners that lead the consortium for the NEPAD e-school demonstration program. The influence of such heavy industry partners will properly give proprietary software a dominating position in this e-school network. The e-school project will influence Ethiopia's SchoolNet project.

The more practical side of this story is what I experienced while I was in Ethiopia. In Addis Ababa I saw many Internet Cafes and small software shops. There were even some small private computer schools. The network at AAU and everywhere else I was in Ethiopia, were slow and severely congested. At AAU the network was only usable in the evenings. In Mekelle the network was always slow and frequently unavailable at the Internet cafe we used. The service seemed more reliable when we connected from Tigray Health Bureau and from Mekelle university, but the connection was significantly slower than in Addis Ababa.

In Addis Ababa the network was more reliable, but was severely congested. At AAU the network was so congested that it was effectively useless in normal working hours. In the evenings the situation was better. After I came back from my first visit to Mekelle I moved from the university guest house to a guest house run by a Norwegian mission agency (*Norsk Luthersk Misjonssamband* (NLM)). There was one computer at NLM guest house connected to the Internet, this connection was faster than the one at AAU in normal working hours.

The AAU network had an extremely strict firewall for both inbound and outbound traffic. The only access to the Internet was through a http proxy, all other communication ports was closed. To get access to the http proxy you had to have a user name and password provided to you by AAU. The communication port for http is port 80, services using other ports was not available. I tried to use other services through the http proxy without avail. If this was not enough AAU also had a web filter filtering out "inappropriate" web pages. Some perfectly legitimate (no porn) web pages in Norwegian was blocked by this filter. I disdain porn so I wouldn't mind actual porn being filtered, but it is better with a little porn getting through than legitimate pages getting filtered. At the NLM guest house I had no such problems. More concerning than the filtering of porn is filtering of web content critical of the Ethiopian government. The Ethiopian government is accused of filtering out critical weblogs at blogspot.com[1].

[1] http://www.worldpress.org/2373.cfm

Figure 7.1: Map of Ethiopia

Chapter 8

The HISP project in Ethiopia

In this chapter I will give an account of my experiences from working in Tigray. Our team were assigned to the health authorities in Tigray. Ethiopia had previously expressed interest in testing DHIS and five regions were selected as pilot sites. The five regions were Addis Ababa, Amhara, Benshangul, Oromiya and Tigray. Tigray is located north in Ethiopia, at the border to Eritrea. As previously mentioned it is from this region the uprising that overthrew the old communist regime grew. Because of this Tigray have much power in the current government. It is in this region the border war between Eritrea and Ethiopia took place.

In total I spent just under four month in Ethiopia. As I have FLOSS as the main subject for my thesis I am going to interpret my experience in the light of FLOSS. My work in Ethiopia was not to ask people about FLOSS and what they knew about it, but to participate in an ongoing FLOSS project, namely *Health Information System Program in Ethiopia* (HISP-ET). I have made no formal interview, but where relevant I will refer to conversation I have had with people in Ethiopia.

As my experience from the field work in Tigray consist of a multitude of situations, words, sights and impressions I am going to choose some themes to use as a skeleton to hang my experiences on. FLOSS is the theme of my masters thesis, so this is one part of the skeleton. My work in Tigray is related to FLOSS in the sense that the software we installed is FLOSS, and that I participated in adapting this software to the needs in Tigray. I will start with some facts about Tigray.

8.1 Tigray Demographics

Tigray region	
Regional capital	Mekelle
Official Language	Tigrinya
President of the Executive Committee	Tsegay Berhe
Population 2005 est.	4.4 mill
Area	est. 80,000km^2
Religion 1994 census	
Christian	96%
Muslim	4%
Ethnic groups 1994 census	
Tigrinyan	95%
Amhara	2.6%

About 83% of the population are farmers. Teff, wheat, and barely are the main crops. Teff is used to make the ubiquitous traditional Ethiopian bread injera. Centuries of erosion, deforestation and overgrazing have left the region with dry and treeless plains, hills and plateaus. Tigray is mountainous, elevation of the region rises from 600-2,700 above sea level, and is one of the richest areas in Ethiopia in mineral resources. A map of Tigray can be seen in figure 8.1.

The Tigray region hold many historical sites. Tigray hold Axum, the seat of the ancient Axumite kingdom. In Axum there is a park of obelisks or Steles dating from as early as the 2nd century BC. Another distinctive feature of Tigray is its rock-hewn churches. This churches are hewn straight out of bare rock. I found it very interesting to hear that the Ark of the Covenant is said to have been brought from the Temple in Jerusalem to Axum. It is said that the son of the Queen of Sheba and Solomon, Menelik, was born in Ethiopia and later lived in Jerusalem. After one year in Jerusalem he was sent back and some of the men sent with him took the Ark of the Covenant. This was brought to Axum where it is to this day, in a chapel guarded by a priest. Only the guardian is allowed to see the ark.

8.2 The Tigray team

I was appointed by Jørn Braa to adapt DHIS to the needs of the Tigray region. In this endeavor I was part of a team with four members. The members were; Solomon, Hirut, Netsanet and myself. All the members of the team except me were Ethiopian nationals. In this section I will give an account of our doings in this region.

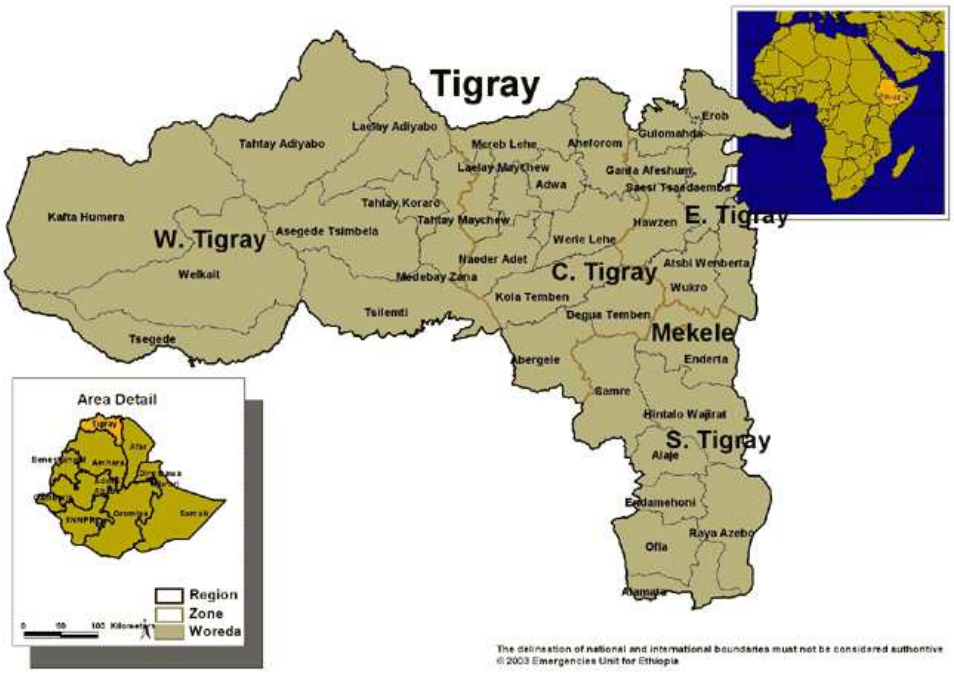

Figure 8.1: Map of Tigray

I came to Ethiopia on Tuesday 6th of July 2004. I came with a open mid, you could say, because I didn't have a very clear idea of what I was supposed to do in Ethiopia. I knew that I was supposed to work with a health statistical system called DHIS, and that I was assigned to a region in Ethiopia called Tigray. At the airport I met Solomon Bishaw, which was the leader of the Tigray team. The other members of the team, Hirut and Netsanet I met later on.

During my first month in Ethiopia we did not do much. Solomon had already been to Mekelle twice before I came. Mekelle is the regional capital of Tigray in which the administrative authority of the health sector in Tigray is situated, the Tigray Health Bureau. He had some sparse contact with the head of the bureau, Theodros, after the first visits, and the agreement was that we should come to start the process of adapting DHIS to the needs of Tigray. Solomon prepared a document on what we were planing to do in Tigray, which he sendt by e-mail.

I, for my part, spend the first month on adjusting my self to Ethiopia on many different levels. I lived in unfamiliar surroundings and in a way I was like a child again and had to learn basic skills from the ground. I will

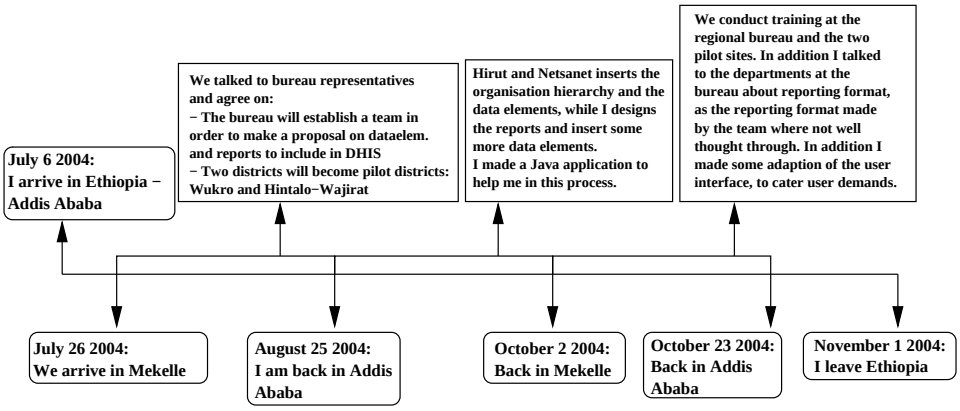

Figure 8.2: What I did in Ethiopia

tell more about this in section number 8.7. As I did not know the DHIS software well, I used this month to learn this software better. Most of my time, however, was spent on familiarising myself with the new surrounding. Everything took time, from buying food to get a visa for the whole four months I was going to stay there.

On Monday the 26th of August we arrived in Mekelle by air. The day after we went to the regional health bureau. We had some previous contact by e-mail and we expected them to be aware of our arrival, but when we came to the bureau there was a sense of confusion. There seemed to me that there had been some shortcoming in the preparation for our visit. The bureau head was not present and there didn't seem to be anybody there that knew about our initiative.

We had four subject we wanted to discuss with the bureau.

1. The importance of a minimal dataset (EDS).
2. The making of a team to establish this data set.
3. Which district that should be a part of the pilot phase of the project.
4. Which resources the bureau could make available to us.

There were a lot of asking around. First we visited a team leader in the *Health Management Information Systems* (HMIS) unit. Most of the discussions were held in Amharic because we decided that it would be better if the bureau staff could express themselves in a language familiar to them. I

don't speak Amharic, so something might have been lost in translation, but from what I could understand from the translation Netsanet gave to me, he was skeptic of our initiative. He said he did not have authority to decide on what we asked for.

Therefore we were lead to an other office, were we met the head of the Evaluation and Surveillance unit. We discussed our plans with him and the team leader in HMIS. They expressed concern about the coming of the Ethiopian new year. They had to finish the yearly report for Tigray, and were therefore rather busy. They thought that it would be better if we came back in September. We had limited time in Ethiopia and could not agree to that. They said that they had been working on a minimal data set which they had sent to the federal authorities for approval. In September the data set would be available. It was decided that we could collect data from their existing computer based health statistical system (See section 8.3). So we spent the rest of the day collecting data from different departments.

From our visit to the different departments we understood that there was a lot of double reporting. Many of the departments in the bureau had its own reports they expected the district to fill inn. The same data was reported multiple times to different departments. There was already at this time a sense of distrust between the *Decease Prevention and Control department* (DPC) and HMIS. The DPC used exclusively their own reports.

It was later decided that we could visit two districts, or as they call them in Ethiopia; weredas. The two districts were Wukro and Hintalo-Wajirat. On the Thursday 29th of July the others on the team went to Hintalo-Wajirat. I stayed behind that day because I had an acing stomach. From what the others told I understood that there was some use of information in the district. They made a list of the ten most common deceases and bought in medicines accordingly.

The next day we traveled to Wukro. This time I felt well enough to come along. We found a mini-buss going to Wukro. The road there had recently been paved and parts of it was still under construction. The construction was done by a Chines firm. According to Solomon there was just a mud road there the last time he was there. It seems that at least some progress are being made in Ethiopia.

After an hour we came to the district health administration. The administrations facilities were good according to Ethiopian standard. We met the head of the administration. The conversation where conducted in Amharic, which left me out. I will give some main points to the conversation as translated to me.

He claimed to be interested in using information for the administration of the districts health sector. The tour he gave us at the administrations facilities confirmed this. We saw a lot of hand made charts and tables. They had made the charts and tables based on raw data and indicators. This show that you don't need computers to manage information. Computers, however, makes it easier (with the right software) to make meaningful information out of raw data, and give better protection against errors in the raw data.

They had some computers at the administration, but they were not used much, except that they used Epi-info to type in information about outpatients. They had one man hired as a computer clerk, but the head was not very pleased with him. The computer clerk had some training in Epi-info, but he found it difficult to use. He also had some beginners training in Java. The next time we came to Wukro, over one month later, I gave him Eclipse and a Sun JDK. We visited the man responsible for the malaria office. He showed us the different reports that the clinics, community workers and hospitals had to fill inn. A community worker are responsible for a variety of tasks in a geographical unit like a village. They are not health workers, but register information about malaria.

After lunch we went to visit the district hospital. There we met the computer clerk from the administration. He introduced us to a lady who was responsible for data collection at the hospital. She had no education for the work, but she had work experience. The data collection at the hospital was exclusively paper based. The lady had a computer installed in her office a couple of weeks back, but she had not started using it. She had no computer experience and the computer only contained Epi-info, which requires computer knowledge beyond the "point and click" level.

The next week I struggled a lot with my stomach and eventually had to go to a clinic to get subscription for some antibiotics. This effectively hindered me to do much work that week. On Monday we spoke to the team leader in HIMS again. He expressed even more of his skepticism. He thought that we, as students, were only interested in our paper and that our initiative would not be sustainable. He didn't want to help us. The next day the others on my team got to speak to the head of the planning department, as I laid in bead with fever. The head of the planning department was much more positive to our initiative. We made an agreement to demonstrate DHIS for the bureau.

On Thursday I felt well enough to participate in the negotiation with the bureau. This day we finally had a chance with speak to the head of the bureau. He sowed himself to be very courteous, and spoke good English. Because of this we held the meeting in English. We explained what DHIS is

and the philosophy behind HISP. He mentioned many of the problems we had noticed. There was a lack of coordination between the departments. Because of this the information flow was like spaghetti. DPC department mistrusted HMIS for managing information. He mentioned that the bureau had a big reporting burden. They had to fill inn many forms for both the federal authorities and NGOs. He mentioned a need for a minimal dataset at the federal level, but he didn't expect that a minimal dataset would be agreed upon for quite some time. He asked us for a milestone plan for our initiative.

The time after our meeting with the bureau head was hectic. We got some problems importing data from epi-info. We decided to convert the datafile format used in epi-info into the format used by DHIS's import/export system. Both are purely text based formats. Solomon used MS Access for this purpose, but didn't manage to import the resulting datafile into DHIS. I, on the other hand, made a python script to automate the conversion. At first I didn't mange to import the file, but when I changed the separation character (the character used to separate columns in the import file) from comma to semicolon it worked. There was an issue with the localisation setting in MS Windows. Different locales use different separation character and this influenced MS Access.

The plan was to present our system on Friday next week. Solomon worked on the milestone plan that the bureau head requested and on the presentation. Hirut typed in the organisational hierarchy for the regional level and for the two districts we planned to use as pilot sites. I worked on importing data and making a presentation of the software. Before Friday Solomon got news about Sundeep Sahay being in Ethiopia. Sundeep is a professor at the University of Oslo and have long experience from HISP in India. He could not come to Mekelle before Saturday. Therefore we asked the bureau to postpone the presentation until Monday next week. Sundeep was a great asset in editing the milestone plan and for the presentation we held at the bureau. It gave our initiative more credibility to have an experienced professor compared to only three master students and a PhD student.

At Monday 16th of August we gave a presentation of HISP for the bureau. The meeting was held at the bureau head's office. There were many people there. The head had obviously called many leaders at different level in the bureau to the meeting. First Sundeep presented HISP with examples from India. Then I demonstrated the functionality of DHIS, based on the data we had imported from epi-info. Solomon closed our presentation by presenting our milestone plan.

After our presentation there were a lot of questions. One question related to whether DHIS had support for capturing community data, not just data

about people visiting health facilities. Sundeep answered that DHIS was not well suited for that kind of application. DHIS is designed for the primary health sector, with clinics, health centers and hospitals etc. There were also some questions about how DHIS did data validation, to answer this I demonstrated some data validation support in DHIS.

Two days later the answer to our proposal came. They had approved it. A team was set from members of the bureau to decide on data elements and report formats. The mandate of the team was to make a proposal on the data elements and reports to be included in DHIS. This proposal was then submitted to a group consisting of leaders from different departments who finally approved the data elements and report formats. Hirut and Netsanet agreed to be our representative during this process. Solomon went back to Addis Ababa because he was soon to go to England for some lectures there. The next week I went back to Addis Ababa too.

I spent a little over one month in Addis Ababa. This time was an important time for me personally, I met a lot of interesting people. In connection with the HISP project we had one tasks; adapt DHIS to support the data elements and reports the bureau team decided on. In other words we could now start the coding related task of the project. Hirut and Netsanet came back to Addis Ababa a little less than two weeks later. They had the data elements and the report formats with them. The details of this work is described in section 8.4.

The 2nd of October Hirut, Netsanet and I came back to Mekelle. This time we had Kalkedan with us. Kalkedan was hired by the HISP Ethiopia project to take care of the work in Tigray after we left. Kalkedan's responsibility would be to give support for DHIS to the health bureau. Kalkedan had some basic computer education from Mekelle University, where he had learned C++ and some VBA and the MS Office suite.

Our plan for this visit was to install the software and conduct training at the bureau and our two pilot districts. As mentioned in section 8.4 the report formats given us by the bureau was not well suited to be used in a computer based system like DHIS. Therefor one of my tasks was to go to the different departments in the bureau to get a better understanding of what they wanted their reports to contain.

The first week we conducted training at the bureau. The first couple of days we talked to different leaders at the bureau, among others the new bureau head. The former head had become a State Minister of Health. The former head of the planning department had stepped into his place. We also discussed with the head of HMIS on how to conduct the training. Because we were only able to make the extended version of DHIS made by Woinshet to

run with the XP version of MS Access there were some issues. The bureau had MS Office 2000 as a standard and was reluctant to install MS Office XP. For training purposes we agreed to install MS Office XP and DHIS on three computers.

The next week Kalkedan and I stayed at the bureau to finish the training while Hirut and Netsanet conducted training in the pilot districts. They first spent the Monday and Tuesday in Hintalo-Wajirat and then the Wednesday in Wukro. Kalkedan and I continued the training until Wednesday at the bureau, and at the same time we talked to people at the bureau about report formats. We finished the training at the bureau with a test that was well received. My goal with the test was not to test what the individual clerk knew, but to let them in cooperation use a broad range of important functionality offered by DHIS. It tested data input and validation, import/export, making reports and doing data analysis.

Because Hirut and Netsanet left Mekelle after Wednesday, Kalkedan and I had to finish the training in Wukro. So on Thursday we drove to Wukro, this time the bureau had given us a car and a driver. That day there was only one man that participated in the training. It was the computer clerk I met last time we were in Wukro. We used a modified version of the test we gave at the bureau. The computer clerk expressed approval for DHIS, especially that data input was done through one interface. Unlike the current system with Epi-info and multiple paper based reports.

All the training material we used at the bureau I had to make myself. I tried to find some training material on the Internet without avail. During the training sessions at the bureau the trainees where divided into groups of two to three people, each group had access to one computer with DHIS installed. At the first day of the training I planned to use a projector to demonstrate the basic use cases. The HMIS unit head said that they preferred to experiment with the software themselves, so that is what we did. The groups tried to perform the basic use cases while we answered questions and guided them when necessary.

The next week Kalkedan and I continued the work on the reports. In connection with this I showed Kalkedan how to make reports, so he could design report for the bureau when I had left. Our visits to the different departments reveled some tension between the departments. I didn't obtain enough knowledge about the inner workings of the bureau to say anything for certain about the reason for the tension. The leader of the Decease prevention and control (DPC) department said that effort had been made to unify the reporting, but without success. DPC did plainly not trust HMIS for information. DPC thought HMIS used too long time to process the report from the districts. DPC have reason to want information quick in or-

der to detect decease outbreaks. They had a number of weekly reports. DPC had need for weekly surveillance data while DHIS version 1.3 was designed for monthly/quarterly routine data.

After I had finished almost all the reports I was presented with a pile of new reports. This reports were recently approved for use in Tigray. I had only a couple of days left in Tigray so I left the work to include this in DHIS to Kalkedan. On Saturday 23th of October I left for Addis Ababa. After a week in Addis Ababa doing nothing of relevance for the project I took an aeroplane back to Norway.

8.3 EPI-info

The health authorities in Tigray was not without computerised support in their information handling. They used a system called Epi-info, both at the bureau and in the districts. This system is developed and maintained by *Centers for Disease Control and Prevention* (CDC). CDC is a operating component under Department of Health and Human Services in the government of United States. EPI-info is in the public domain, so you can basically do what you want with it. They currently develop and maintain a Windows version of the software, but a DOS version is also available.

In Tigray they still used the DOS version, but Windows versions were also in use. To get an better understanding of what DHIS had to 'compete' against I am going to do a comparison between Epi-info and DHIS in this section.

This is the description of the Windows version from epi-info's web site. The older DOS version together with the *Geographical Information System* (GIS)-tool EPIMAP have the same functionality, but is more difficult to use.

> Epi Info™ is a public domain software package designed for the global community of public health practitioners and researchers. It provides for easy form and database construction, data entry, and analysis with epidemiologic statistics, maps, and graphs. The primary applications within EpiInfo are (CDCP 2005):
>
> **MakeView** A program for creating forms and questionnaires which automatically creates a database.
>
> **Enter** A program for using the forms and questionnaires created in MakeView to enter data into the database.

Analysis A program for producing statistical analyses of data, report output and graphs.

EpiMap A program for creating GIS maps and overlaying survey data on to them.

EpiReport A tool that allows the user to combine Analysis output, Enter data and any data contained in Access or SQL Server and present it in a professional format. The generated reports can be saved as HTML files for easy distribution or web publishing.

Here you can see that Epi-info is not very different from DHIS when it comes to data gathering and analysis. The DHIS equivalent to MakeView and Enter are the data elements you configure DHIS to use. DHIS offer the possibilities of printing out questionnaires too. By using pivot tables, GIS and the report module of DHIS you get a similar functionality to Analysis, EpiMap and EpiReport programs of DHIS.

The differences are, however, significant. The most important difference are based in the different problems they try to solve. Epi-info are made with decease surveillance in mind. The data that need to be gathered are different for each possible outbreak that need to be monitored. Therefore Epi-info is designed to make it easy to make forms, input data, design report and do data analysis. Epi-info is a excellent tool for gathering a lot of data and do data analysis on them, and then to move on to another project.

DHIS on the other hand are designed with the primary health system of, specifically, South Africa in mind. One leading thought behind DHIS have been that the primary health units like clinics and hospitals gathers data and report it to the district, the district reports to the regions and so on. From DHIS version 1.4 any unit at any level in the organisational hierarchy can be the reporting unit. Data in DHIS are also connected to a specific time period, a month or a quarter.

The notion that data exist within an organisational hierarchy and belong to a time period leads to a big difference between DHIS and Epi-info. In DHIS data are connected with an organisational unit, the organisational unit exist within an organisational hierarchy. Epi-info have no specific notion of an organisational hierarchy or reporting period. You can have an input field in a form that specify the organisational unit it belongs to and which time period it belongs to, but the data are not easily aggregated across to health units, districts, region or time period.

It is interesting to note that both systems is in a sense FLOSS. EPI-info is public-domain software, you can get the source code if you request it. DHIS

is licensed with a FLOSS license. Both relies on non-FLOSS software components, however.

8.4 Adapting DHIS for Tigray

Hirut did the greater part of inserting the data elements. The number of data elements were past 1100, but a number of them were similar to the ones in Addis Ababa which Woinshet, the head of the HISP project in Ethiopia, had already typed into DHIS. We based our work on what Woinshet had done for Addis Ababa. The data elements and reports that the team from the bureau decided to be included in DHIS were different in a number of ways. The reports that were supposed to be sent to the federal authorities were the same, but most of the reports used internally in Tigray were different from those in Addis Ababa.

The months before my arrival Woinshet put a great effort into adapting DHIS to the requirements of Addis Ababa. Many of these requirements were the same in other regions we worked. The most important of these requirements was that they wanted the system to handle data based on *The International Classification of Deceases* (ICD). ICD is basically a number assigned to deceases according to a system of classification. In Ethiopia this codes were used for in-patients and out-patients to specify which diagnosis the patients got. In addition the gender and the age group the patient belonged to was recorded. Deaths were recorded in the same manner. If this kind of data were supposed to be included as normal data elements in DHIS each combination of ICD codes, gender, age group and whether it was in-patients, out-patients or death should have it own data element the number of data elements would pass 16,000 in Tigray.

DHIS is designed with a minimal data set in mind. Preferably below 100 data elements. In Tigray they wanted close to 1,200 data elements in addition to the ICD data. In Addis Ababa they wanted a similar number of data elements. Because of this Woinshet made a extension to DHIS to handle ICD based data. This extension was hard coded into DHIS version 1.3.0.17, therefore this was the version we used in Ethiopia. This extension being hard coded meant that it would have to be reimplemented for each new version of DHIS. It would probably be possible to make something similar to a patch that could easily be patched into new versions of DHIS. The fact that DHIS is made in MS Access and don't have source code in the traditional sense makes this more difficult. MS Access is a RAD tool and stores its data and code in a binary format.

While Hirut inserted the organisational structure of Tigray and most of the

data elements, I inserted the quarterly data elements and started on the reports. The reports given to us did not fit well to be implemented in a computer based system. To me it seamed like the bureau team had only gathered as many reports they could find and given it to me. They had not given thought to what data elements they wanted and how this data could be presented in reports. The list of data elements they had given us where also lacking data element necessary for a number of reports.

The reports they had given us where not made with computers in mind. Some reports required regular data to be given in a comment/note field. Some of the reports where weekly based, mostly from DPC, which DHIS 1.3 don't support. What made things even more difficult where the fact that a number of reports where in Amharic or Tigrinian. The other people at my team understood Amharic, but none of us understood Tigrinian. Because of this challenges I was only able to complete the the quarterly reports, that is the reports on Tuberculosis and Leprosy.

Because I was unable to finish the reports with the information I had, I decided to postpone this task until I could talk to the departments at the bureau. To make use of the time I had left in Addis Ababa, before we were to go to Mekelle, I started on a small Java application. This application automated time consuming tasks like ordering the data elements and inserting test data. The test data were needed in order to test the reports. This application I have described in an other paper.

During our training at the bureau the following changes were proposed:

1. Automatically insert the values from the previous row into the new row. This pertain to the graphical interface into the ICD extension made by Woinshet. This would save a lot of repetitive selection values in the ICD input form.

2. Amharic names on the months.

3. Have separate drop-down boxes for month and year

4. In Tigray they used a separate set of decease codes in clinics. The ICD code is used by hospitals and health centers, but clinics have their own codes. They wanted the ICD codes to show up when you selected a hospital or a health center and the clinic codes when you selected a clinic.

5. The order of input fields where different between the selection of inpatients, outpatients and deaths in the ICD input form. This could be considered a bug.

I implemented all this request except number 2 and 3. I didn't find any easy way to change the month names in MS Access into Amharic names. The input clerks doing the typing could speak sufficient English to understand this. English is used as a administrative language alongside Amharic in Ethiopia. Requirement 3 i dropped because it would require more than changing the user interface. Months and years are coupled in the data model of DHIS. I could possibly had conceived a way to do this without changing anything, but the user interface. I didn't deem it that important, however.

8.5 Problems with the DHIS software

In Ethiopia we made extensions to the DHIS software that effectively made it incompatible with the South African version. This extension was made because the Ethiopian authorities demanded the use of ICD based data elements for reporting. If each data element were to be inserted as a standard data element it would become several thousand data elements, many of which would be rarely used. In Tigray it would become past 16,000 data elements. The data set chosen in Tigray was already far from a minimal dataset, with in the vicinity 1,200 data elements.

To effectively handle ICD based data we had to make changes to the core of DHIS. The regular data elements is like a hash table with a data element type and a value. The ICD data is more like a sparse matrix, in a health facility only a few combination of gender, age and ICD-code is used. To solve this there was basically two options; ether hard code the extension into the DHIS Access application or make a visually separate application feeding on the DHIS database. In Ethiopia it was chosen to hard code the necessary extensions into the DHIS application. Because MS Access is a RAD tool it was not possible make the extensions without making it incompatible with the South African version. This effectively made the Ethiopian DHIS a fork. It was decided to hard code the extensions because it would make DHIS more acceptable for the intended users. One user we trained expressed that he liked that he could do all the data input through one interface.

We could have sent the extensions to the South African team, but Ethiopia was the only country which required support for the ICD code based input. Because the extension was hard coded it was not possible to choose not to have it. If DHIS 1.x was extensible, you could choose to include only required aspects of the system. This is, however, not so. DHIS 1.x is relatively easy adaptable to new contexts in that you can easy insert data elements

and organisational units and relatively easy make reports and graphs of the data. It is not extensible in the sense that you can choose to include external modules.

We also experienced the problems with incompatibilities between different versions of MS Access and with incompatibility caused by using different language locales. Even if the South African DHIS was tested for MS Office 97, 2000 and XP the extensions made in Ethiopia made DHIS only work on the XP version of MS Access. Officials at the Tigray Health Bureau expressed discontent with this because they had MS Office 2000 as standard at the bureau.

The installation procedure we had to use was also cumbersome. First we had to install the South African DHIS using its installer, then we had to copy our version of the DHIS application and the data file over the original. We wanted to make an installer for the Ethiopian DHIS, but the South African team used an expensive proprietary program to make the installer. If we had the source code for the installer available we could have made some small tweaks to make it install the Ethiopian DHIS application and data file.

8.6 Getting support from the HISP community

On three occasions during the process of adapting DHIS to Tigray we sought help from the HISP community. On the first occasion we had problems importing data from EPI-info into our prototype. On the second occasion I had problems implementing GIS support in DHIS. On the third occasion I sought information about how to make an install script incorporation the changes we made in Ethiopia. All the attempt at getting help was limited by the slow and unreliable network in Mekelle, and to a lesser extent in Addis Ababa. The only means of communication was through e-mails.

The problems we had with importing data from EPI-info, I have already mentioned. We basically found out of this problem ourselves with little useful information from the HISP network. In Mekelle the network was frequently unavailable, which in a way made e-mail just as interactive as regular mail, it went days between each time we checked our e-mail and then we had to dig our way through spam. It could take minutes to open each e-mail in a webmail client.

At one point during my stay in Ethiopia I planned to implement GIS in DHIS. I had no idea on how to get about this. I experimented some with ArcExplorer and looked at the content of related files. I had a hard time

figuring out what to do to implement this. I found no relevant information on the Internet. I was in Addis Ababa where the network was usable when I tried this. I had some e-mail correspondence with the HISP community, but did not get sufficient information to understand how to implement GIS. I really missed a step-by-step howto and explanations about the technologies used. Documentation relating to DHIS and related software was altogether rater limited. There was also a lack of communication channels for this kind of support questions. The end of the story was that I decided not to spend time on this as I had other things to do.

The question I had about making an installer for the Tigray version of DHIS was quickly settled as I got the message that I had to have some expensive proprietary software to change the install script. In other words I couldn't reasonably make an install script. The most helpful support we got from the HISP community came through the visit of Sundeep Sahay.

8.7 Being the farench/faranji

In Ethiopia they have a word they use at foreign looking people, especially white people. Actually it is two words which seem to have the same meaning. One of these word I read about in my guide book to Ethiopia, this word was faranji (Briggs 2003). Faranji can be translated to foreigner, not necessarily a white foreigner. The other word was farench. This word, I was told, came from an Ethiopian general that didn't manage to pronounce French right. This later came to describe white people. In the time the socialist regime reigned and foreigners from Cuba were more common, kids in the streets of Addis Ababa were shouting "cuba". After the fall of the socialist regime this seam to have changed to "farench".

In this section I will give a short account of how I experienced being a "farench" in Ethiopia. It is always differences in the culture, big or small, between different countries and even between different cities. The biggest barrier between people of different cultures is not as much in the way of behaviour as it is in the language. The people I worked with in Ethiopia spoke English and I therefore had no greater difficulty relating to them. I don't think it would be more difficult to work with French people for instance. The difficulties I faced where mostly related to language, contaminated food and to too much attention.

The language problems were evident in our dealings with the Tigray Health Bureau. A lot of discussions were conducted in Amharic and I had to ask for a summary of the discussion from my colleagues. The three people I

worked with in Tigray all were from Ethiopia and therefore found it easier to speak in Amharic, leaving me out. Often the conversations at the lunch and dinner tables were done in Amharic. The others spoke in English if I said something, but the conversation almost always automatically went over to Amharic until I said something again. It is very boring looking at people speaking in a language you don't understand.

What put most strain on my stay in Ethiopia where the almost two weeks I where having serious stomach problems. I was expecting to get some problems, but I got more than I bargained for. I where amazed at how week I felt laying in bed with fever, even getting up from bed where a challenge. I experienced what many have experienced before me, that I am not invincible. It is easy to be proud and self reliant when you are healthy and things go your way. I suppose I experienced what Job in the Bible experienced, but on a much smaller scale. I were grateful when the fever passed and I again could focus on our work. Through it all my relationship with God and the Narnia books (written by C. S. Lewis) helped me through the day.

Being a white man in a poor African country I could only expect a lot of attention. White people from the developed world coming to Ethiopia are immeasurably more wealthy than the beggars on the streets. I can't really blame them for asking me for money. It is just that there were so many of them. When I walked from the university guest house to our office in the Siddis Kilo branch of the university there were between five to ten people asking me for money. I found it difficult to just ignore this people, but after some time I realised that I just had to.

The beggars were men and women, old people and children. The children were very determined and wouldn't give up. Many children sold paper napkins, chewing gum and similar things. They have to do this in order to afford going to school, I was told. Some aid organisations provided napkins for children to sell. It is more dignifying to sell things than to beg.

The begging I could deal with and was understandable. The frequent shouting from children and even some adults were more annoying. I often heard someone shout "farench" or "faranji" after me. It also annoyed me that they took a much higher price from me than from my Ethiopian colleagues at hotels. Considering how much wealthier I am even as a student, than most Ethiopian people the lower price for Ethiopian nationals don't seem unreasonable. On the other hand Ethiopian people using hotels is not poor, and it doesn't give an impression of hospitality.

There is not much violent crime in Ethiopia, so I felt quite safe. Petty theft is, however, not uncommon to face, especially if you are white. One time

I was walking in the streets of Addis Ababa one man threw him selves at my feet, in order to distract me I guess. I noticed two men approaching from behind on my left and right side. I managed to get out of the situation without loosing my money, thankfully. This was the first time in my life someone have tried to rob me.

It might seem that I make a pretty grave picture of my stay in Ethiopia, but I am very grateful for the chance a got to be there. I met many interesting people. Both Ethiopians and, after the Norwegian summer holidays where over, I met many Norwegians. My colleagues from HISP where easy to get along with, and I got the honor of being invited to a wedding. Coffee is like small encouragements through the day, so I was happy that Ethiopia is a coffee country. Coffee is said to be originating in Ethiopia, and espresso like coffee with lots of sugar is available everywhere. Hot milk with coffee where also common, and became one of my favorites. Finally I consoled my self with the fact that even if my surroundings were unfamiliar to me, working with a computer was not.

Chapter 9

FLOSS in Ethiopia

During my stay in Ethiopia I hoped to discover how the situation for FLOSS was there. Most of my time was spent on adapting DHIS, dealing with Tigray health bureau and being sick, so I didn't get time to do real research about FLOSS. FLOSS in Ethiopia are still too small for statistical research to be interesting. The thing I learned about FLOSS in Ethiopia I learned from searching the Internet and talking to people. What I have found I will present in this chapter.

9.1 Economic argument for FLOSS

In this section I will use some simple calculations to show the cost of license fees for a country like Ethiopia. This is to show that even if the license fee in many cases are a small part of the *Total Cost of Ownership* (TCO) it is significant for a developing country. TCO is a measure that aspire to give an indication of what the cost of a program is through all its life span.

According to CIA World Factbook there were 75.000 Internet users in Ethiopia in 2003. According to a 2003 estimate (Encyclopedia Britannica) there lived over 65 million people in Ethiopia. This give a Internet users rate at 0.11% of the population.

The GDP/capita in Ethiopia is $800 according to World FactBook. At Amazon you have to pay $319.99 (5.sep 2005) for MS Office 2003 standard version. For MS Windows XP Home edition you have to pay $179.99. This means that an Ethiopian have to work 7.5 month of GDP/Capita without eating to buy the common MS Windows and MS Office combo. In Norway we have a GDP/capita of $40000, which means we have to work 0.15 month

(4.5 days) for the same combo, without eating. For Windows 2003 server (5 client) cost $788.99 which becomes almost one year in GDP/capita.

Compared to buying a new PC which cost in the vicinity of $600 - $1200, the software is rather expensive. MS Windows is often included in the package when you buy a new PC. Because of deals Microsoft makes with PC manufactures, MS Windows is sold at a significant lower price. According to Ove Arntzen at Dell in Norway the price for Windows XP Home bundled with their PC's is $50. This implies that the operating system constitute from 4% to 8% of the PC' cost. If you include MS Office 2003 the proprietary software constitute from 62% to 21% of the PC's cost. Considering that you have to have applications for the operating system and computer to be of any use, it is reasonable to compare the total price of the software running on the PC with the PC's cost.

It is often difficult to buy PC's without MS Windows because Microsoft have Original Equipment Manufacturer (OEM) agreements with many PC manufacturers. Manufacturers who assemble brand marked PC's get a better deal on Microsoft licenses the more they choose to affiliate with Microsoft. The more PC's with windows a manufacturer sell, the better deal the manufacturer get with Microsoft.

FLOSS licenses do not restrict how the software should be distributed and permits distributors to charge for the service of distribution. All the small computer shop I saw in Ethiopia sold pirate copies of proprietary software. If this shops were to get hold of good FLOSS software they could sell Cd's with this software legally. A German organisation called *Relevantive* (wwww.relevantive.de) visited Ethiopia in 2004 and planned to initiate an information centre for FLOSS (http://www.relevantive.de/osic.html). This centre was going to give support, information and hand out Cd's with FLOSS software. This project is currently stalled.

9.2 The TRIPS agreement

One man I met in Addis Ababa said that MS Window was open source because you could get it for free. He meant he could get a pirated binary copy of Windows free of charge. This man had not really understood what FLOSS is, but he had a point. As a rule software in Ethiopia are pirated. I can't blame them. Why should a poor country like Ethiopia give money to already far to wealthy western companies? It is not like the companies are actually loosing money, they just receive less profit.

Many companies do argue that they loose money, and based on the assumption that everyone[1] would buy the software if they had to pay, they calculate the loss. Many of those who obtain pirated software would never buy the software for the real price. This estimated loss is part of the reason the *Agreement on Trade-Related Aspects of Intellectual Property Rights* (TRIPS) (World Trade Organisation 1994) have been made. The TRIPS agreement was made through the *World Trade Organisation* (WTO) and took effect in 1995. This agreement was made in order to get laws about IP more similar between countries. The least developed countries were given 11 years to implement the agreement, meaning that Ethiopia should have implemented it by now. Software piracy in Ethiopia are epidemic, so I wonder how Ethiopia are going to enforce IP laws concerning software.

I find it hard to be very sympathetic towards the TRIPS agreement. After all, who have the most intellectual property to protect, the developed or the developing world? No matter what you may think of it, however, it creates an even better incentive for Ethiopia to start using FLOSS. There is reason to believe that software piracy can strengthen Microsoft's position in the developing world. Through software piracy Microsoft's products get into widespread use, and thereby familiarising people with its products.

9.3 Political support for FLOSS

Despite the ambitious aspirations of the Ethiopian government for leapfrogging Ethiopia into the information age the awareness off and knowledge about FLOSS is low. The attitude of the Ethiopian government can be illustrated by this statement by Meles Zenawi, quoted in Guardian (Cross 2005):

> "Our position is determined by the fact that proprietary suppliers have the money to provide initial support," he says. "To implement open source needs a minimum of training and at the moment we don't have that. In five or 10 years time, we will be in a position to choose."

To check what kind of impact this policy decision has had on the choices of web servers, both the operating system running the web server and the web server it self, I have done an analysis of the network in Ethiopia in section 9.7. From this analysis I found only one web server using FLOSS software, of the 37 web servers considered.

[1] Everyone they estimate to have a pirated copy of a proprietary software.

Relevantive, in their travel to Ethiopia, interviewed the Head of Computer Science Department at AAU, Dr Dawit Bekele.

> He said that the main drawback for open source software is the common opinion; "If we share, we loose", regarding the sharing of code by open sourcing it. The Ethiopian government invests heavily into software development, but the ministries and administrations keep the code closed and to their own. After some years it is obsolete and has to be rewritten – more than often from scratch

(Horstmann 2004).

9.4 FLOSS usage

My experience from Ethiopia confirms that the awareness of FLOSS is low even though the knowledge and usage of ICT is on the rising. I was curious about whether software shops in Ethiopia sold any FLOSS software, so I went into all the software shop I could find. Most of this shops sold unlicensed copies of proprietary software alongside some software that at least came in a box, many which seemed second hand. Even though it is perfectly legal to sell Cd's with open source software I did not see any shop trying to do this.

The previously mentioned organisation Relevantive was in Ethiopia in connection with a planned project with the following aims in Ethiopia (Muehlig and Horstmann 2004):

- Provide technological infrastructure (i.e. computers and related equipment)
- Provide knowledge on open source technologies.
- Initiate a project that may work as a blueprint for similar future projects in Ethiopia or other African countries.
- Understand and gather knowledge on culture specific usage of technology.

Some representatives from Relevantive traveled to Ethiopia in April 2004 in connection with the project's evaluation phase. In the report from this

travel, a man named Daniel Yacob was mentioned in connection with open source. Daniel Yacob is director of the *Ge'ez Frontier Foundation* (GFF) (http://www.geez.org/) and is very active in the Amharic translation and localisation of open source software. He is working on an Amharic translation of the Gnome desktop environment. Daniel Yacob mentioned three considerations about open source in Ethiopia (Horstmann 2004).

1. FLOSS Software being free of cost was no advantage, as most of the software in use is either provided by development aid or pirate copies.
2. Developing FLOSS Software voluntarily in one's spare time would find no supporters, as the need for paid work is overwhelming.
3. One of the major advantages of Linux turned out to be its security concept and invulnerability towards viruses. As the local knowledge regarding computer/network security is very low and - because of low bandwidth – the stakes for downloading the latest patches and anti-virus signatures are very high, keeping computers free of exploits is highly valuated.

As might be expected I saw very little use of FLOSS in Ethiopia. The exceptions are our software DHIS, which had not actually come to use yet, and the public domain Epi-info. At AAU I saw that they used the FLOSS web proxy software Squid. The *United Nations Mission in Ethiopia and Eritrea* (UNMEE) (http://www.unmeeonline.org/) uses FLOSS and is conscious about it. The Ethiopian newspaper *The Reporter* has an on-line edition hosted on a Linux server and using Apache and PHP, but this host is located outside Ethiopia. The Reporter has been a useful source for me in writing this thesis. Other than this I have seen little evidence of FLOSS usage in Ethiopia.

9.5 Ethiopian FLOSS organisations

Most FLOSS project are international, with contributions from actors from different countries. To find projects that can be said to "belong" to a specific country can be difficult, even more for Ethiopia with its overall limited participation in FLOSS. An obvious exception to this are language related projects. Other exceptions are organisation promoting FLOSS for a specific country or region of the world, and user groups like a *Linux User Group* (LUG).

In search for user groups relating to FLOSS I looked at the directory of LUGs listed at http://www.linux.org/ and found none. At the Linux Counter

(http://lugww.counter.li.org/) I found one. I checked this at the 7th of August 2006, when I checked this just some moths earlier I found none, so things are happening in Ethiopia. Computers are still far from becoming a tool for hobbyists in Ethiopia, but there are encouraging developments. The use of computers is mostly limited to businesses and the government. Given the feeble support for FLOSS by the government it is no surprise that the use of FLOSS in Ethiopia is still limited. However, I found one Ethiopian FLOSS organisation working on localisation to Ethiopian languages and the recently formed Ethiopian Free & Open Source Software Network (http://www.efossnet.org/) which was formed in February 2005.

Given Ethiopia's current political system of ethnic federalism, where each region can decide on their own working language, the support for multiple languages and translations between them is important. Microsoft have only recently started to localise MS Windows and MS Office to Amharic. Given Microsoft's feeble interest in localising the most spoken language in Ethiopia it is a long shot to think they will do it for Oromo or Tigrinya. With FLOSS software the Ethiopian government can pay developers and translators to do the job and pay in local currency.

The issue of localisation is what the previously mentioned GFF has as its mandate. GFF have made a number of small tools for viewing and editing the Ge'ez script for Windows and Unix systems. Interestingly it has done work to localise Latex and Perl. GFF have even done work on making proposals for the Unicode standard for the Ge'ez script. From the source files released by GFF I have read, it seems like the work has almost exclusively been done by Daniel Yacob. On SourceForge I found a project called LibEth (http://libeth.sourceforge.net/) which is related to GFF.

At the time being there is little organised activity on behalf of FLOSS in Ethiopia, despite recent encouraging developments. Africa as a whole is poorly represented in the FLOSS community. Because the awareness of FLOSS is rising steeply and many begin to see FLOSS as a mean to bridge the digital divide, this is now changing. An organisation called *Free and Open Source Software Foundation for Africa* (FOSSFA) (http://www.fossfa.net/) have been founded to promote the use of FLOSS in Africa. FOSSFA has it origin from a ICT workshop in Addis Ababa held in November 2002. An other organisation seeking to promote FLOSS in Africa is Open Source Africa (http://www.opensourceafrica.org/).

9.6 Participation of Ethiopia in FLOSS

Ethiopia faces the same challenges as other developing countries when it comes to participation in the FLOSS community. In centres of higher learning in Ethiopia there are people with sufficient knowledge to contribute to FLOSS. Internet access is congested because of a low bandwidth international connection, but this will improve in the near future if the plan to connect to the EASSy cable comes true.

The current participation in FLOSS project by Ethiopians are related to internationalisation and localisation. The work that is supported by the GFF are examples of such. Another project is being lead by Dr. Dawit Bekele at AAU. Dr. Dawit together with a team of three other participants have made a prototype localisation of OpenCMS into Amharic. OpenCMS is a *Content Management System* (CMS) written in Java. The previously mentioned Daniel Yacob is an interesting person in the Ethiopian localisation community. Daniel Yacob is the responsible person for localisation of Gnome and Open Office, as well as a number of localisation software hosted on the GFF website. He has also involved himself in the localisation of the Ubuntu Linux distribution.

A project which is not software related, but is through information sharing in the spirit of FLOSS, is Wikipedia. Wikipedia is a free encyclopedia. All the articles in this encyclopedia is made and edited by voluntary participation. Anybody can by default make or edit articles in this encyclopedia. Mechanism are in place to censor out illicit content and behaviour, and to mark articles according to its quality. This extremely low barrier to editing articles have worked well to provide a lot of information, but a critical sense is needed. Then again a critical sense is an asset even when dealing with more authoritative sources. The Wikipedia project aims to have articles in all languages, and have facilitated this. The Amharic Wikipedia have 274 articles as of April 2006. The Wikipedia for other languages used in Ethiopia have barely any content at this time.

9.7 Analysis of the network in Ethiopia

Because of curiosity and because I wanted to see the usage of open source operating systems and web servers on Ethiopian hosts, I have done some investigation on host in the `et` DNS domain and on IP addresses owned by ETC. I will first describe how I searched for the host and what programs I used to do this, then I will present the result.

When I started this search I had no clue to what IP addresses belonged to Ethiopia. IP addresses is not organised according to geography, but more or less randomly distributed by Internet Registrars associated with *Internet Assigned Number Authority* (IANA). IANA assigns available IP addresses to *Regional Internet Registry* (RIR). RIR assigns IP addresses to *Local Internet Registry* (LIR) or *National Internet Registry* (NIR) which again assigns IP addresses to Internet service providers.

To find the IP-range owned by ETC i decided to search for all the hosts registered in the `et` DNS domain. ETC is the only ISP in Ethiopia, so if I could find the IP range owned by ETC I would find all IP addresses in Ethiopia. The DNS domain `et` is also owned by ETC. All the hosts in the `et` domain is not necessarily located in Ethiopia, but I am more likely to find hosts located in Ethiopia in this domain. My first thought where to query the DNS server for the `et` domain. I found the DNS server for this domain by querying the WHOIS database. The different RIR's have WHOIS servers with a database over assigned domain names, IP-addresses and various other information. I used the program `whois` witch searches a number of WHOIS servers. By using this I found the following DNS servers for the `et` domain.

DNS servers for the **et** domain

```
Nameserver Information:
    Nameserver: ns1.gip.net.
    IP Address: 204.59.144.222
    Nameserver: ns1.telecom.net.et.
    IP Address: 213.55.64.36
    Nameserver: ns2.gip.net.
    IP Address: 204.59.1.222
    Nameserver: ns2.telecom.net.et.
    IP Address: 213.55.64.38
    Nameserver: ns3.gip.net.
    IP Address: 204.59.64.222
```

DNS uses port 53 for communication and by searching this nameservers I found that all the nameservers in the gip.net domain were up and running. The nameservers in the telecom.net.et domain had a firewall that filtered out port 53. When I at a later time checked the telecom.net.et servers they where not even running. I concluded that the servers in the gip.net domain where the ones in use, with ns1.gip.net as the primary server.

Now I had the primary DNS server for Ethiopia so now I tried to find all the sub-domains registered in this server. This can be done with something called zone transfer. The purpose of zone transfers is to copy the content of a DNS server database to be used by a secondary server. Zone transfers are often only allowed for hosts within a specified IP-address range. This where the case for the aforementioned servers. Because I couldn't find any other way to get all the records of a DNS server I dropped this strategy.

The next strategy I thought of where to search Google for all `et` domains. Google offers domain specific searches. This way I could find most of the hosts in the `et` domain running a web server. To automate this I made a script that made a search query to Google and parsed the result. The parsing captured all sub-domains of `et` in the links found in the search result. Because many host names can be served by the same physical host I looked up the IP address for the host names.

When I first did this at 13th of September I found 89 host names on 11 physical host. When I did this the day after I found 157 host names on 21 physical hosts. Many of this hosts where not located in Ethiopia. I searched the WHOIS database for the different IP-addresses in the result and found that the range `213.55.64.0 - 213.55.95.255` belonged to ETC. There are over 8000 IP-addresses in this range, which is a small number for a whole country. Using a *ping sweep* on this address range on 13th of September I found 895 host to be up. A ping sweep sends a small message to every IP-address in the range and checks for replies. I used a program called `nmap` for this. Normally an ICMP Echo message is used for pinging, but firewalls often filter this messages out. nmap uses various techniques to circumvent firewalls. Most of the previously mentioned hosts is probably only clients with no server software running on them.

I repeated the ping sweep on 15th of September. This time I also checked if port 80 where open on the running host I found. Port 80 is the standard http port used by web servers. I wanted to know how many of the host I found where running a web server. Of the 1162 host I found that 42 had an open port 80. I combined this with the hosts in the ETC IP address range, which I found through the Google search. I got a total of 46 web hosts. Then I did a *operating system scan* and *version scan* on this 46 addresses. An operating system scan tries to guess the operating system running on the host and a version scan tries to guess the version of the server software running on the host. With this data I can get an estimate of the usage of FLOSS for web servers.

Of the 46 IP addresses I decided to scan for operating system and web server version, 39 were up. This 39 hosts will form the basis of my study of the usage of FLOSS programs in the Ethiopian web server marked. Of the 39 hosts that were found to be up I found information about 37 of them (See table 9.1). The Cisco IOS and MS ISA web servers are not web servers meant to present content to the Internet, but used to configure the other services running on it. The other services can be fire-walling, routing, VPN or other kind of infrastructure related services.

	Operating System				
Web Server	Windows	GNU/Linux	Solaris	Cisco IOS	Unknown
Apache		1			
MS IIS	17				
SunONE			3		
Netscape			1		
Cisco IOS				11	
MS ISA	1				
Unknown			2		1

Table 9.1: Operating systems and web servers used in Ethiopia

Chapter 10

Development of a Plug-in Framework for DHIS 2

In this chapter I am going to describe the second case study I did as part of my master thesis research. Compared to the case study I conducted in Ethiopia this is a smaller case study, but a relevant study non the less. This study is more technology focused as it is a practitioner study where I act as a FLOSS programmer. This case has longer time span compared to my work in Tigray, but my involvement in this case has only been part time. I conducted this case study in parallel with writing this thesis, and doing other tasks relevant to my studies. Through this case I hoped to learn about participating in a FLOSS project on a practical level, at the same time as I wanted a chance to do more programming.

In Ethiopia I worked with DHIS 1.3 and there was ongoing efforts to implement the next major release in the 1.x strain of DHIS, version 1.4, in South Africa. DHIS 1.4 is an architectural remake of the DHIS database, and to a lesser extent the user interface. DHIS has over the years been developed layer upon layer to incorporate changes demanded by users. The code increasingly came to resemble a spaghetti like mess, which was hard to maintain and few really understood. The pre-DHIS 1.4 architecture also had to be changed to enable important new features. Some of the assumptions implemented in the core in the previous releases was specific to South Africa at the time of implementation, this had to be delegated to the flexible exterior of the application. Some of the limitations in DHIS 1.3 was:

- The organisational hierarchy was limited to five levels.
- Data entry was only possible for the lowest level in the hierarchy (the facility level).

- It was only possible to collect monthly data in the main module and quarterly data in the tuberculosis and leprosy (TB) module.

DHIS 1.4 is still implemented in MS Access, but the architectural remake removes the previously mentioned limitations. DHIS 1.4 introduces many larger and smaller improvements, in addition to removing this limitations. One of the improvements is that DHIS 1.4 makes it possible to use a different *Database Management System* (DBMS) than the one MS Access uses. At the time of writing it is possible to transfer structures and data to a MS SQL Server, and there are plans to make transfer tools to MySQL and Oracle too. The user interface is run in MS Access, but it uses another DBMS in the back end. DHIS 1.4 also introduces the concept of *data sets*. Data sets groups together a number of data elements which is given a reporting interval. A data set can for instance be data related to malaria reported on a weekly basis, or finances reported on a monthly basis.

Development of DHIS 1.4 has been going on for some time. At the time of writing DHIS 1.4 is released, but some features are still not implemented. DHIS 1.4 should be regarded as a "bridging" version to DHIS 2. DHIS 2 is the version I have concerned me with, and is the topic of this case study. DHIS 2 is a total reimplementation of DHIS 1.4 using only FLOSS components, releasing DHIS from a dependency on specific versions of MS Office.

10.1 Why the need for a reimplementation?

There are several shortcoming in the DHIS 1.x line which is linked to the limitations of MS Access and the style of development it supports. This limitations is the primary reason it was decided to make a total reimplementation of DHIS. I will explain several of the limitations in this section.

MS Access is a *Rapid Application Development* (RAD) tool where the focus is more on facilitating fast application development than it is on making the application maintainable. As a consequence DHIS 1.x is not very modularised. The user interface, consisting of forms and reports, is tightly integrated with VBA scripts and database tables. The DHIS 1.x strain is primarily developed by one small team in South Africa. Because of the "onion architecture" of the software only the development team leader, Calle Hedberg, understood the software in depth. The term "onion architecture" reflects the fact that the software was step vise expanded and improved according to user demands. This changes was made layer upon layer, creating tight dependencies between different part of the software. MS Access

naturally lend itself to this style of programming. The design of DHIS 1.x have not been done in splendid isolation, but the development is and has been done by one small team.

DHIS 1.4 fixed some of the architectural problems the previous releases had. This version is, however, still limited by the MS Access platform. MS Access is not made with a open distributed development process in mind. MS Access is as a RAD tool primarily developed to support application development by one programmer, or a small team of closely cooperating programmers. MS Access is in other words designed to support small scale cathedral-style development. To harvest the collective capabilities of the HISP network it was decided to base the development of DHIS 2 on a community model. To facilitate this the core of DHIS had to be built on a modular base.

A serious limitation in MS Access is the maximum size of the MS Access database, a single database file is limited to 2GB. At the central level in South Africa the database size was closing in on this limit. The MS Access database is not designed to handle many simultaneous requests and therefore do not scale well according to the number of users. Neither to MS Access scale well according to the size of the database. This is mostly solved by the possibility to transfer the database to a better DBMS in DHIS 1.4. Still DHIS 1.4 relies on the MS Access user interface which looks simple and out-dated. The user interface is clearly improved in DHIS 1.4 compared to 1.3, but is still limited by MS Access. A fancy user interface is not important for the functionality of the system, but is important to the overall user experience.

Turning our attention to the internal architecture of the DHIS 1.x it should be noted that MS Access is built around a two tire architecture. There is a user interface tire with forms and reports and there is a database tire with tables and queries. The business logic is spread into both the user interface tire and the database tire. The current best practice for database applications is to use a three tire architecture which clearly separate between user interface, business logic and the database. This architecture is easier to change and extend, and is better for distributed development.

The DHIS 1.x strain is a stand alone desktop application and the only way to transfer data between each instance of the application is through exporting and importing of data files. For larger offices with several people doing data input and analysis it requires some effort to export and import data between the different DHIS installations. It would be easier to have a central database accessible by LAN. To support different use scenarios ranging from a single installation to a corporate LAN it was decided to web-enable DHIS.

The cost for MS Office and MS Windows licenses was not an issue in South Africa when DHIS was first implemented, because MS Office and MS Windows was already in widespread use. The MS Windows and MS Office combo is commonly available in developing countries, with or without a license, so the cost savings argument have up until now not been a strong argument. This can change as Microsoft shows more interest in emerging markets, and because of international agreements like the TRIPS agreement.

Basing an application that is supposed to be used by many different users in different contexts on MS Access gives several challenges. MS Access is clearly not designed for this. Even the basic boolean values; true and false, have different representation in different localisations of MS Access. The update life-cycle of MS Office and MS Windows makes it necessary to support a matrix of MS Office, MS Windows and language combinations. We experienced this in Ethiopia with the column separator issue and the issue with the ICD extension to DHIS, making DHIS only compatible with the XP version of MS Access.

10.2 The community model in DHIS 2

The development process used to develop DHIS 2 is more like many other larger scale FLOSS projects. With distributed development between more or less independent actors. For this to work a community has to be built around DHIS 2. In this section I am going to describe how this is done for DHIS 2.

The HISP community is the most important asset for the development of DHIS 2. There are many different peoples within this community (or network if you will) ranging from ICT professionals to health workers, and with people from both high-cost and low-cost countries. HISP has close ties to the academic community and a lot of contribution is sought from this community. Contribution is also sought through paying ICT professionals in low-income countries. Through seeking contributions from this two sources an application can be built with very little monetary cost. Master students to not get any pay. PhD students gets their pay from the state or other sources of scholarship. ICT professionals from low-income countries like Vietnam to not demand much money.

At the University of Oslo, which is the central node for DHIS 2 development, a course is held called *Open Source Software development* (INF5750). The practical assignments of this course is to participate in the DHIS 2 development. Most of the students taking the course are not seasoned Java

programmers, and the student spend a lot of time familiarising themselves with the frameworks used in the DHIS 2 development. The course is credited for one third of a semester, so there is a limit to what the students are able to do. Some of the students might catch a more long term interest and continue after the completion of the course, through writing a master thesis relevant for DHIS 2 and later on a voluntary basis. This is not unlike other project where there is a relatively small core and many who make small contributions. In the following paragraphs I will look into who the major contributors to the DHIS 2 core are.

Judging by the number of commits to the DHIS 2 Subversion repository there are three people who have contributed more than any others. All this people are Norwegian and are or has recently been master students at the University of Oslo. The three top commiters according to the number of commits are responsible for 50% of the total number of commits. Looking at the number of commits is a poor metric for measuring contributions, one commit can possible be more significant than a hundred commits.

To be better able to judge the value of the contributions I will use the size of the difference caused by a commit as a metric. When changes are committed to the repository the difference between the current version and the committed version is stored. The size of the difference is the number of characters in the diff log. This is a quite good metric because the addition or change of text files create large diff logs, but addition or change of binary files creates small diff logs. Source files are typically text files. By this metric only one of the three previous top committers are still among the top three. They are still among the top six committers, however. Among the new three top committers there are two Norwegian master students and one Vietnamese. According to this metric there are six committers that stands out. The six top committers stands for 79% of the total size of all the diff logs. All of these except one are Norwegian master students. In total there are 73 committers. See Table 10.1.

Nationality	Role[a]	Commits	Size of Commits[b]
Norwegian	Master student	13	28863222
Norwegian	Master student	450	10054784
Vietnamese	Hired	55	7461401
Norwegian	Master student	15	7243223
Norwegian	Master student	212	5515440
Norwegian	Master student	259	5482552

[a] Reason for contributing
[b] In characters

Table 10.1: The top six contributors to the DHIS 2 core (31st of August 2006)

There are of cause other ways to contribute to DHIS 2 than to commit code. Contribution of documentation and support is just as important, and so is the maintenance of the HISP web site. The efforts made to test and configure DHIS 2 out in the field is not shown by the Subversion commit statistics, neither. It seams clear, however, that it is mostly Norwegian master students who contribute to the DHIS 2 core. HISP have hired four developers in Vietnam, who previously was master students. One of these came in third place according to my Subversion commit statics. The managerial role in the project is taken by two PhD students. In addition to the work done on the DHIS 2 core there is work going on in India, Vietnam and Ethiopia to test and configure DHIS 2. DHIS 2 has very limited cooperation with other faculties at the University of Oslo and limited contributions from the health systems where DHIS is used.

For the development of DHIS 2 we have a web site[1] and commonly used services like version control, wiki and issue management system. An overview of the collaborative services used can be seen in Table 10.2. Communication

Service	Software	License
Wiki & weblog	Confluence	FLOSS project/Non-profit license[a]
Issue management system	Jira	FLOSS project/Non-profit license
Version control system	Subversion	BSD-style license
Mailing list	Mailman	GPL
Build system	Maven 2	Apache License

[a]Special non-FLOSS license provided by Atlassian

Table 10.2: Collaborative services used in the DHIS 2 development

is mostly done through the e-mail list. More direct communication through the use of IM, telephone and talking face to face is also quite common. It is difficult to track the direct communication, so it is difficult to say how much communication is done in this way. The Wiki and Weblog are only used by a few.

DHIS 2 is currently licensed under a permissive BSD style license. This is done in the hope that it will make DHIS 2 more acceptable in business settings, and widen the use of DHIS 2.

10.3 My role in the project

I started to participate in the HISP network at a time when talks about implementing DHIS on a totally new platform had recently started. The at-

[1]http://www.hisp.info/

titudes in the Norwegian node of HISP, which were going to be the central node in DHIS 2 development, was clearly in favor of implementing DHIS 2 using Java. There was some early talks about using a *Java 2 Enterprise Edition* (J2EE) framework and *Enterprise Java Beans* (EJB), but this was abandoned because J2EE and especially EJB has a notorious reputation for complexity. Not long before I left for Ethiopia it was more or less decided that DHIS 2 should be implemented using Java, but using lightweight FLOSS frameworks in place of J2EE. The Spring framework was a good contender, so I was given some literature about Spring that I brought to Ethiopia. Curious as I was about this framework I spent some time in Ethiopia learning this. The helper application I made in connection with the adaption of DHIS 1.3 in Ethiopia was made using this framework.

When I came back from Ethiopia I was interested in spending some time working with the DHIS 2 development. At the University of Oslo they had just started to give the DHIS 2 development course so I singed up. I also participated somewhat in the planning of this course and the major thing that concerned me was that DHIS 2 should be extensible. This was the reason for forming a group to look into the possibility for making a plug-in framework for DHIS 2. The reason for my concern was the issues relating to the ICD codes in Ethiopia. The lack of extensibility in the core of DHIS 1.x had mandated a choice between making a separate application feeding on the DHIS database or making a fork of DHIS.

The group formed consisted of me and an other master student, Ole Petter Aasen. Aasen took the university course from another city in Norway named Lillehammer while I was located in Oslo. Aasen was in Lillehammer because he had a job there. We met face to face before the course started and only one time during the course. Most of our interaction was done through IM, and a lesser extent e-mail. We also kept some wiki pages about our plug-in module and had a project in the issue management system, which was a course requirement. After the INF5750 course had come to an end I was essentially on my own in experimenting with the creation of a plug-in framework.

10.4 What motivated me to participate

I had basically four things that motivated me to participate in the DHIS 2 development. I wanted to learn to be a better Java programmer, and learn the interesting frameworks and tools used in the development of DHIS 2. I wanted to actively participate in a FLOSS project, to learn how to act and communicate in such a setting. I wanted to work on an actual application,

not just some useless application made to learn Java. And last I needed the credit points that the course gave me. Course or no course I would have participated in DHIS 2 development, anyhow.

10.5 Making the application extensible

Inspired by my experience with the ICD extension in Ethiopia, which was hard coded into DHIS version 1.3.0.17, I was determined to make DHIS 2 sufficiently extensible to be able to make the ICD extension without making a fork. On the planning session we had about the upcoming course INF5750 I said that I wanted to look into different ways of making DHIS 2 extensible. In this section I am going to present my experiences from this task.

Initially I had a plan to make DHIS 2 extensible through the use of plug-ins. Plug-ins are encapsulated pieces of software designed to be plugged into a mother application. This plug-ins provide additional features to the mother application. Plug-ins are usually placed in one or more specified directories where the plug-ins are automatically discovered at application start-up. The plug-ins can also be specified in an configuration file or a start-up script. I and Aasen, my coworker during the duration of the course, decided to start looking for existing plug-in frameworks. The mother application needs to be designed with plug-ins in mind, and a plug-in framework provides the sockets where plug-ins can be plugged into. Our initial plan was to find a plug-in framework and find out how this can be used with the mother application to provide the necessary extensibility to be able to make an ICD plug-in.

We looked into the following plug-in frameworks. This frameworks are pure plug-in frameworks. Pure plug-in frameworks are designed to have just a minimal mother application which basically just starts up the plug-in framework, discover and start plug-ins. All the functionality is provided by the plug-ins. These plug-ins will also provide sockets which other applications can plug into. These extension sockets are commonly called extension points. Other plug-ins can subscribe to an extension point. The plug-in providing the extension point is responsible for providing the necessary features and resources to the plug-ins subscribing to the extension point. Such an application naturally needs a minimum number of plug-ins to be able to operate.

- *Java Plug-in Framework* (JPF)
- Emersion (Web-app framework based on JPF)

- Platonos
- Knopflerfish OSGI
- Eclipse

For some of the frameworks like JPF you can make the mother application as large as you want, it is not necessary to make a pure plug-in application from it. JPF is a minimal plug-in framework, it only provides the minimal functionality necessary. Eclipse is an application in its own right, providing many useful extension points. None of these plug-in frameworks, except Emersion, was made with web applications in mind. Emersion is designed for web applications and uses JPF as its plug-in framework.

Emersion did not really fit our need, but we decided to use it as a starting point to make our own server side plug-in framework. The DHIS 2 application was at this time in its infancy of development, so we concluded that it would be too difficult to build a plug-in framework into the evolutionary prototype being developed. This would have required us to convince all the other developers to build their modules using unfamiliar concepts like extensions points. Besides we had not yet proved that it was feasible or desirable to make DHIS 2 using a pure server side plug-in framework. For this reasons we decided to make a throwaway prototype as a proof of concept. We gave this prototype the unimaginative name plug-in-demo.

For reasons I will explain in the following paragraphs it became a real challenge to make plug-in-demo. The development of plug-in-demo was more than enough work for the course. We never came around to implementing extension points in the real DHIS 2 application, and we could plainly forget about making an ICD plug-in. I continued to work on plug-in-demo after the course had ended.

One of the major issues that created complexity for the plug-in-demo was the need to embed a servlet container. A servlet container is a web server built to support servlets, which provide dynamic content using the Java language. JPF needed to be booted before the servlet container, and needed to run as long as the servlet container was running. The alternative would be to start JPF each time a user accessed a web page, which is a huge overhead that would make the application unbearably slow.

Embedding a servlet container within a JPF context created class loader issues which was difficult to debug . In Java the class loader is responsible for loading classes when a class instance is requested. The class loader needs to have access to the description of the classes which the application needs. This created problems for plug-in-demo because JPF and the servlet

container were running within different class loader contexts. Even if JPF had access to the description of a class it did not mean that the servlet container had access. We managed to resolve this, but it was hard to debug this problems and could possibly create problems in the future.

A sever problem with using JPF was that the application became difficult to test. The creation of unit tests is considered a important practise for the creation of stable applications. An unit test is a script that test a single unit of code, like a class. The test should be done in isolation, the dependencies the class has on other classes should be satisfied without calling the other classes. For example if a class depend on data from a database the test script should not call the database, but provide the data by other means. It was really difficult to satisfy the dependency a class had on JPF. I wasted a lot of time trying to find a solution to this without avail.

An equally important testing issue are integration tests. This test the interaction between different parts of an application. This kind of tests uses the actual classes and databases used by the application. In the JPF context this means testing the plug-ins when they are actually working together. To do this the mother application and all the plug-ins has to be deployed to their assigned places, and the JPF framework has to be started. This I had to do manually by building and deploying all the plug-ins in the plug-in-demo application, and then start the application. If the application did not start I had too look in the log files created by the application to try an locate the bug, fix the bug and build and deploy the application again. This is time consuming.

In addition to the embedding and testing issues there was issues with integrating the other frameworks used by DHIS 2, and also plug-in-demo, with JPF. I spent a lot of time finding and implementing ways for JPF to integrate with Spring. To this I found solutions I am quite happy with. Spring is like JPF an all-encompassing framework affecting the architecture of the application, and require the developer to get familiar with unfamiliar concepts.

The forth and final challenge I met in building plug-in-demo was the challenge of using Maven to build the application. It was already decided to use Maven as build tool for DHIS 2. The responsibility of a build tools is, among other things, to compile all the source files, link the application to all of its dependencies and install or deploy the application. Maven also offer support for executing automatic tests. To build an application based on JPF I had to make a plug-in to Maven. First we used Maven 1, and I made a reasonably functioning plug-in to that version. Later on the Maven project finished a total remake of the Maven application called Maven 2. The DHIS 2 project started to use Maven 2. It was more difficult to make a JPF plug-in for Maven 2 than it had been for Maven 1. The design of Maven 2 had not

considered applications of the JPF kind. I managed to make a JPF plug-in to Maven 2, but it was only a really ugly hack.

Reflecting on how difficult it was for me to make this framework and the thought of having to decide on where to place extension points, I decided that it would create more complexity using a pure plug-in framework than it would give value in better extensibility. Neither did I find a satisfactory solution to the Maven 2 build issue. There are other approaches for making an application extensible, which is not so all-encompassing.

There exists many different way of making an application extensible. Some extensibility is achieved through *Object Oriented Programming* (OOP) and even more is achieved through an *Inversion of Control* (IoC) container which is the primary function of Spring. OOP offers the possibility of extending classes and implementing interfaces. If a class is doing almost what you want it to, but not quite, you can extend the class to make the necessary changes. The new class can act as its parent class through polymorphism. It is also possible to make a new implementation of an interface. A class usually interacts with other classes and when a class is instantiated it also needs access to instances of the classes its depends on. An IoC container has the job of inserting this dependencies. By using an IoC container it is easy to change which implementation of an interface is handed to a class, or to insert a sub-class of the depended class. Common to this forms of extensions is that they require some configuration file or source file changes in the core application.

The existing DHIS 2 application uses OOP and Spring to facilitate extensibility. In addition it provides an *Application Programming Interface* (API). This is a handy way to provide extensibility by providing a way for other application to interact with DHIS 2. Other applications can talk to the business layer of DHIS 2. Having an API do not in itself solve the problem of having visually separate extension to DHIS. An API can be a handy way for plug-ins to talk to the mother application, however. An API is not as likely to change as its implementation.

An other way to achieve extensibility is through hooks. Hooks can be implemented as a global array of function pointers or object references. The extension insert its own function or object into this array, and this array is iterated through at some point in the core application. I have seen the concept of hooks being used in other web applications, a hook is similar to an extension point. Hooks are more light weight than a plug-in framework, therefore it is no need to embed a servlet container. Hooks are easier to test because they do not require any framework, and the integration and build problems are history for the same reason. Each time a user request a web page which access a part of the application providing hooks, all the

extensions using this hook has to be included. For the mother application to include the extension some minimal configuration or source file changes has to be made. Automatic discovery is also possible. The discovery algorithm would have to be run every time a user access web pages with hooks, however. Each individual hook will most likely be more difficult to implement, and it will probably require more effort to include an extension of this sort into the mother application. Figuring out which hooks to provide will be just as difficult as figuring out which extension pints to provide.

Hopefully the local adaptations in the upcoming version of DHIS can be done using extensions, in place of forking or visually separate applications feeding on the DHIS database. This remains to be seen.

10.6 Interaction with other DHIS 2 developers

In FLOSS projects it is an advantage to talk a lot. By talking a lot I could have made the other DHIS 2 developers more familiar with what I was doing, and I could have gotten handy tips from the other developers. During the development of plug-in-demo I have not participated much in the virtual DHIS 2 community. My interaction with other DHIS 2 developers have been very limited. The development of plug-in-demo was so different from the DHIS 2 development that I did not expect the other developers to be interested.

I and Aasen worked together while the course lasted. After that time I have been on my own. I participated in the planning of the course, as already mentioned. In the summer after the course had ended a session was held where the results of the course contributions were presented an evaluated. I presented plug-in-demo at this session. I had the impression that the other participants in this session did not really understand the concepts embedded in plug-in-demo. Perhaps they did not understand the concepts because I held a poor presentation, but I interpreted into a confirmation of my suspicion that a pure plug-in framework for a web application was too complex.

10.7 Interaction with projects we depended on

I interacted more with the projects we depended on to build plug-in-demo than I interacted with other developers of DHIS 2. I had interaction with

two external projects; JPF and Maven. In this section I will describe my interactions with this two projects.

The need to interact with the JPF developer came when I tried to integrate JPF and Spring. JPF is a small project with just one developer. I downloaded the source code of JPF and tried to figure out how to do the integration. It was not possible to do the integration without making significant changes. I could not do the integration by extending classes or implementing interfaces, in other words I could only change JPF not extend it. I saw three routes forward; I could ask the JPF developers to incorporate my code into the official release by providing a patch, I could make a fork of JPF and thereby missing out of further releases of JPF or I could ask the JPF developer to make some architectural changes to JPF to facilitate my extension.

I sent an e-mail to the JPF developer where I explained what I wanted to do. The JPF developer asked me to send him the code I was working on. I gave him some suggestions on how to make JPF extensible enough for my Spring integration code. I sent him the integration code I had made and two different patches to JPF, one for each of the strategies I suggested. The JPF developer took my suggestions into consideration and came up with an architecture quite different from what I had suggested. He had made quite large architectural changes. Using this new architecture I found it quite easy to integrate Spring and JPF by extending JPF. Having a concrete problem I wanted to solve, and some code and suggestions helped me to convince the JPF developer to change the JPF architecture. The JPF developer is also the developer of Emersion.

The DHIS 2 project changed from using Maven 1 as its build tool into using Maven 2. Maven 2 have deep differences from Maven 1, and uses a completely different architecture for plug-ins. For this reason I had to make a JPF plug-in for Maven 2 completely from scratch. I had made a JPF plug-in for Maven 1, but this could not be used. I read documentation on the Maven web site on how to make plug-ins for Maven 2 and started to code a JPF plug-in. Once again I came into architectural limitations. The Maven 2 built system had not been designed with application based on a pure plug-in framework in mind. I spent a lot of time reading documentation, and I downloaded the Maven 2 source code. I spent a lot of time searching in the Maven 2 source three to figure out how to make my JPF plug-in.

To get some help on how I could make my JPF plug-in I sent an e-mail to the Maven 2 user e-mail list. I got no answer on my e-mail. I asked in the wrong place, I had very peculiar needs for my JPF plug-in. After spending many ours on coding my JPF plug-in, reading the Maven 2 source code and testing possible solutions I found a way to bend Maven 2 into building my plug-in-demo prototype. It was an ugly hack, but it was the best I could

do based on the existing architecture of Maven 2. I made some thoughts on how to change Maven 2 into supporting a JPF plug-in, and sent in two feature requests using the issue management system of Maven 2. Now I got some answers and I explained my need. This two feature requests are planned to be implemented by the 2.1 release of Maven 2. Having a concrete need again helped me in convincing that the changes was necessary.

Part V

Discussion and Conclusion

Chapter 11

Discussion

As described in the introduction of this thesis my research objectives was to explore FLOSS and how FLOSS are benefiting and can benefit Ethiopia and developing countries in general, and how FLOSS are benefiting and can benefit HISP. This discussion is framed within the grand theory of structuration theory and a social informatics perspective on technology. First I discuss the FLOSS and Ethiopia connection and then I discuss the FLOSS and HISP connection. Last I will make some general theoretical considerations.

11.1 FLOSS and Ethiopia

In this section I will first make some reflection on the effective use of the Internet and then of FLOSS in Ethiopia. I will use the term *effective use* as described in section 2.3.1. That is, I will look at the physical, digital, human and social resources available for the effective use of the Internet and FLOSS. This discussions have to be seen in the light of section 5.7. Last I will discuss the HISP effort in Ethiopia in the light of my experience from Tigray, as well as in the light of my discussions about the effective use of the Internet and FLOSS.

11.1.1 Effective use of the Internet

Currently the physical access to the Internet is limited. The WAN inside Ethiopia is being actively developed to remedy this, and by connecting to

the EASSy cable Ethiopia will get a much better connection to the international Internet. This is just the physical resource needed for the effective use of the Internet. There is also a need for relevant content accessible through the Internet. Ethiopia have quite a few government web pages, but those I have seen have not been very interesting. More interesting are Ethiopian newspapers, and weblogs and discussion forums where Ethiopians and expatriate Ethiopians discuss current political issues. Most of these digital resources are only available if you know English. There are some web pages available in Amharic, but for other local languages in Ethiopia there are barely any content.

The government of Ethiopia have high aspirations and there are quite a few students enrolling into ICT education. But the PC penetration rate is low and enrollment into first level education are still low. I don't think it is a daring prediction to say that even if the higher strata of the Ethiopian society will catch up with the developed world there will be a significant digital divide within Ethiopia. This is typical for developing countries. Developing countries constantly have to catch up, leaving the lower strata of society behind. A question that is still open is how open the public's access to the Internet will be. There are some disturbing signs of government censorship in the form of filtering out weblogs with government critical opinions.

Internet is the blood vein of the large majority of FLOSS projects, it is through this medium cooperation and contributions to FLOSS project are being made, and it is through the Internet you can get access to FLOSS software. Ethiopia's high aspirations for building infrastructure for Internet connectivity can help in this regard. During my stay in Ethiopia the network was severely congested and slow. The time required to download a Linux distribution would be prohibitively high. The current international Internet connection have very limited bandwidth. To be able to effectively participate in an international FLOSS community, adequate bandwidth is important. The connection with the EASSy cable will improve this. The stability of the improved network and whether increased use will congest the network remains to bee seen.

The lack of Internet access can prove a major obstacle to the acceptance of FLOSS among the computer literate people of Ethiopia, as piracy copies of proprietary software is much more widely available.

11.1.2 Effective use of FLOSS

Even if work is being done to increase the capacity of the Ethiopian network this is an improvement only accessible to the elite in Ethiopia. This was also the case in the beginning in USA and Europe until the middle of the nineties. One big difference, however, is the higher general education level and wealth in USA and Europe. This gave a ground for a hobbyist culture around computers. It is little reason to believe that this will be a widespread trend in Ethiopia. It is much more difficult to get a PC for private use and people generally have more pressing needs. To be able to participate in the FLOSS virtual community you have to have three things, provided you are interested in participating; access to the Internet, adequate computer knowledge and time to spend. It is safe to guess that there is not many people in Ethiopia who have this three things. There is limited ground for a hobbyist community in Ethiopia, computers are still prohibitively expensive relative to the average income. It is only in universities, colleges, government bureaus and in the small computer business sector it is likely for anybody to use and contribute to FLOSS. Ethiopia is likely to benefit most from FLOSS in the education sector, and the broader government sector.

Among the educated elites in the universities and colleges there should indeed be grounds for usage of and participation in FLOSS, if the students are made aware of FLOSS and given the opportunity to experiment with software. Using thin-client networks like those promoted by Ndiyo and remote administration, it seem feasible to use FLOSS in schools as part of the e-school program and SchoolNet. This require the education of a group of expert network administrators who are competent in FLOSS software, with special emphasis on Linux and the Unix class of operating systems. At the federal level there is a need for a small group of experts responsible for support and for making helpful documentation, howtos and tutorials. In each region groups responsible for going to the physical location of each school to install and maintain the networks can get help from the core group. In each school a person with basic computer knowledge can be responsible, as this person can phone a regional expert or get help through the Internet. Most maintenance can be done remotely by the regional groups. If there is a need for making changes to the source code of a particular software, in order to make it useful in Ethiopia, the core group with help from students, faculty and professional developers can make the necessary changes. This changes can be contributed back to the FLOSS project maintaining the software.

The limited participation in the FLOSS community by Ethiopia is predominantly focused on translation of software packages. This is a natural place

to start. In Ethiopia with its federal system where every region can decide on the working language of the government and schools, the possibility to translate software into many different languages is a great advantage. If the government is willing to sponsor such projects it will give ICT educated people in Ethiopia jobs and save foreign capital, and giving Ethiopia long term advantages in the form of giving people computer access in local languages, in the form of making Ethiopia independent of ICT vendors and in the form of building local ICT capacity.

The building of institutional support for FLOSS have just recently started in Ethiopia, through the formation of efossnet.org and an Ethiopian LUG, in additions to the efforts by Dawit Bekele and Daniel Yacob. GFF also come into this picture, which is mostly an effort by Yacob to localise FLOSS software. Relevantive had an idea for an information center for FLOSS. By having such centers where it is possible to get FLOSS software and support it will be much easier to effectively use FLOSS. You cannot get FLOSS software from the regular computer marked in Ethiopia. This efforts are limited to the small minority of computer literate people in Ethiopia, but I think it is a step in the right direction.

Prime Minister Meles Zenawi have expressed the opinion that there is a need for a minimal level of training to implement FLOSS solutions, and that Ethiopia have to wait five to ten years before they can choose. I do agree that FLOSS solutions, like all solutions, need a minimal level of education, but I think it is better to start right now to build the necessary capacity. Ethiopia have a chance to choose now when it comes to university and college education. The university curriculum should be reviewed and a narrow focus on proprietary technologies should be removed. My colleague in Tigray, Kalkedan, was only familiar with Microsoft technology after education in Mekelle University. It is also possible to start an education program to build the capacity to implement and maintain LANs in Ethiopian schools, it is not necessary to rely on foreign capacity to do this. Maintaining LANs in schools is an ongoing effort which should not be done in one huge effort and later laid to rust.

For Ethiopia to benefit from and contribute to FLOSS there needs to be a critical mass of people with computer knowledge and familiarity with FLOSS. This is best addressed through the higher education sector. FLOSS has its background from academic institutions because it was there people with the necessary interest, knowledge, equipment and time were found. This is even truer for Ethiopia since the general population is much poorer. The Ethiopian government should also play a major role by sponsoring translation efforts and building capacity to support the effective use of the planned SchoolNet and WeredaNet programs. If this capacity is not rele-

vant for a job in the developed world it is only an advantage that will limit "brain-drain" from Ethiopia.

MS Office and MS Windows was available on all computers I saw in Ethiopia, with or without a license. Software piracy is common in Ethiopia and you can get expensive proprietary software for just a few dollars. The lower cost of FLOSS is currently not an argument for the regular private computer user. The government and business sectors usually need to be more concerned about having licenses for their software, especially if they are to comply with the TRIPS agreement.

11.1.3 HISP, Tigray and Ethiopia

The planned improvements of the Ethiopian WAN will make it easier for the Ethiopian HISP node to collaborate in the HISP network. It will also make it easier for the HISP people in the field to tap into the overall HISP network. The awareness of FLOSS is on the rising in Ethiopia. HISP-Ethiopia have the opportunity to show that FLOSS can be feasible in Ethiopia.

I did not manage to make a usable plug-in framework for DHIS 2 within my time constrains, but DHIS 2 still is much more modularised than DHIS 1.x. This will avoid the problem of making an incompatible fork to support ICD codes. Now the necessary adaptions done to the DHIS core can be sent upstream to the central HISP development team and incorporated in the main DHIS releases, with less likelihood of the changes making problems for the other users of DHIS. DHIS 2 is neither dependent on the client having specific versions of MS Office and MS Windows, all the necessary software can be bundled on a CD. Neither is it necessary with a costly proprietary program to make an installer. This solves the problems we had in Tigray with installation.

The HISP effort in Ethiopia can be a model for how it is possible for Ethiopia to benefit from FLOSS. HISP had, during my stay there, efforts going on in five regions. We had a team in each region implementing DHIS in the region and in some pilot districts. If this is possible for HISP, it is also possible for the education sector in Ethiopia. Communication between the different HISP teams was limited, a better computer network and a web page with documentation and an e-mail list would improve this. We could not improve the network, but we could have made a web page and e-mail list. Everyone in Ethiopia was busy with what they where doing in the regions and had no time to build a central node in Ethiopia. At the time of writing HISP Ethiopia has built a web site (http://www.aau.edu.et/faculties/dis/site/hisp/).

In Tigray we had to "compete" against an installed base. There already existed a computerised system used for data capture and analysis, namely EPI-info. As noted in the comparison between EPI-info and DHIS in section 8.3 there are important design differences between this two systems. EPI-info is designed for surveying possible decease outbreaks and is good for non-routine data collection where the data that need to be collected varies between each case. DHIS on the other hand is designed to gather routine data from the primary health system.

I will say that DHIS in Tigray is best suited for routine data. DHIS will make reporting and data analysis easier as it offers an unified system for routine data collection. Data can easily be exported, sent and imported at the unit above. Data analysis can easily be done for any number of month and aggregated at any level in the hierarchy. If you want data for a district the data from the primary health units in that districts are aggregated together. Epi-info on the other hand can be better for non-routine data collection which the DPC department might need to make. Epi-info is designed with decease prevention and control in mind. DHIS 1.3 do not support weekly reporting intervals used by the DPC department, but later versions do. We only adapted DHIS 1.3 in Tigray, so EPI-info is better for such reports at the current time. DHIS and EPI-info fulfill different needs and are not mutually exclusive.

The DHIS core implements structure and processes common among routine health information systems. DHIS is designed to be relatively easy to adapt to different health systems by facilitating the process of technology translation. By isolating many complex implementation issues in the stable core a lot of the development and maintenance burden is shared with HISP and other countries using DHIS. The development of DHIS is a community process which makes it more likely that the software will stay in sync with changing demands, as opposed to an in-house developed IS only relevant for a specific country. By using the FLOSS licensed DHIS each country do not have to reinvent the "wheel", get a "wheel" as a donation or pay somebody to reinvent the "wheel".

The Ethiopian HISP node have been relative active in DHIS 2 development. The ICD extension have been incorporated into DHIS 1.4, and work to implement an ICD extension for DHIS 2 is being done. Tigray have not started to use DHIS 1.3 for routing data collection in the region as we hoped for, but the Tigray health bureau is interested in testing DHIS 2.

11.2 FLOSS and HISP

A key principle for HISP is that DHIS should be licensed with a FLOSS license. DHIS 1.x is in fact licensed under the GNU LGPL. This is not because HISP is affiliated with the hacker culture or the FLOSS community. The decision to make HISP FLOSS is like the decision to make DHIS 1.x in MS Access, based on pragmatic reasons. This reasons and the similarities and differences between how DHIS is developed compared to the community model of development common for many FLOSS project, is what I will explore in this section.

DHIS being licensed under a FLOSS license gives clear advantages to the user. Unlike when negotiating to make a deal with a proprietary vendor the health authorities do not incur any financial risk by testing DHIS. HISP will not take the health authorities to court for not having enough licenses or for breaking contracts agreements relating to DHIS. The user can copy the software to as many computer as they wish, and if the need should arise they can even modify it. Modifying DHIS can make further updates of DHIS difficult if the changes breaks compatibility with the official branch of DHIS. Modifying DHIS in this way will effectively make a fork. Users outside the HISP network of students and developers will rarely make changes of this sort.

In the process of translating DHIS to a new context, developers sometimes need to look into the source code to make necessary changes. If DHIS was proprietary software the developers working on translating DHIS to a new country would have to sign NDAs, and this would have to happen within the legal framework of each country. HISP would have to hire lawyers. DHIS being FLOSS makes it possible for the people doing the adoptions of DHIS to a new country to make rapid changes based on user demands, making DHIS more acceptable to potential users. FLOSS is a prerequisite for informal collaboration across national and organisational boundaries, with a minimum of legal obstacles. The decision to base a FLOSS program like DHIS 1.x on a proprietary platform like MS Office, might seem a bit contradictory. Both the decision to license DHIS 1.x with a FLOSS license and the decision on build it using MS Access was based on pragmatic consideration seeming obvious at the time.

11.2.1 HISP and the conventional FLOSS community

The HISP community is predominantly made up of health workers and academics and is not affiliated with the hacker culture. It is important to

note that the overall goal of HISP is to empower the poor and marginalised. In this regard FLOSS is seen as a mean rather than an end in it self. FLOSS, however, lends itself naturally to a project with a goal to use IS as a tool for empowerment. Free software being, as it usually is, free as in "gratis" serves the poor, and free software being free as in "freedom" serves the marginalised. FLOSS serves HISP well, especially because HISP is not a profit seeking enterprise, and is based on what in the FLOSS context is called a community model with many more or less independent actors.

Unlike Linux which is based on the POSIX standard it is very hard to give an exact requirement specification for DHIS, so the participation of health workers are essential. Health workers are users and cannot be expected to contribute directly with bug fixes and patches. Health workers have to get guidance formulating what they want, and in what is technically reasonable. This put more challenge on the design of the system. There is a clear separation of the actual users of the systems and the core programmers making the system. Between the actual users and the core programmers of DHIS are the people who do the translation of DHIS, this people configure the system and do other forms of adaptations of the software. The translators plays a mediating role between the actual users and the core programmers. Within the HISP network the translators are the closest we come to user-programmers. The translators do more than translating language. The translators translate all aspects of the software that are necessary to make it useful in a particular context.

Members in other FLOSS communities tend to be more technically knowledgeable. In the later years FLOSS have crawled out of the hacker community and the academic computer science communities. Hackers have intrinsic interest in exploring programmable systems and can cope with software that is not particularly user friendly, and can change the software themselves if they are not happy with it. For enterprise ISs like DHIS with an user base consisting of people who do not care about programming and the inner workings of a computer, the actual users are not programmer. DHIS is not unique in this regard, FLOSS is used in many different kinds of enterprise systems. It make sense to base a new systems on proved FLOSS libraries, like the Spring framework and WebWork in the Java context. In this context the users of the FLOSS libraries becomes the user-programmers, not the users of the enterprise system. In the DHIS 2 context there is a group of core programmers working with DHIS 2. They base the development on a number of FLOSS libraries to make the development of DHIS 2 easier. DHIS 2 is a system that is meant to be used in many different context, so in addition to the core developers we have the translators. The core developers develop the stable core, and the translators adapt the flexible exterior. The actual users of the system do not have

any programming skills, so for the translators the social perspective is most important. An illustration of this can be seen in Figure 11.1.

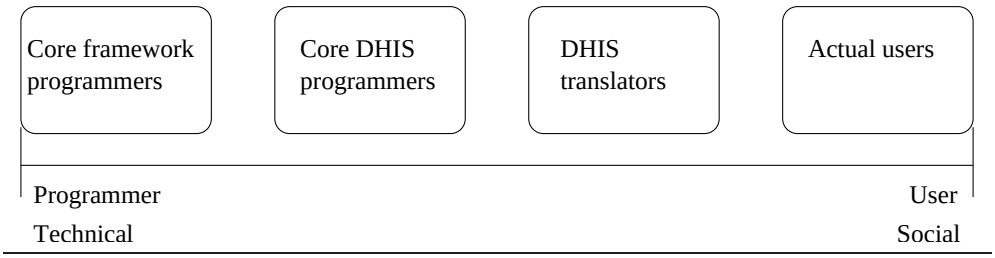

Figure 11.1: The user-programmers within the DHIS context

Many of the academics in the HISP community are people with no passion for programming. This people concern themselves more with the social than the technical aspects of IS, and are more likely to act as translators interacting with the actual users of the system. I worked as a translator in Tigray, but because I have a more technical inclination I concentrated more on the DHIS software than the organisational context. Even within the different boxes in Figure 11.1 the different actors can have a varying emphasis on the social or on the technical aspects.

In general there are very few people from developing countries participating in FLOSS development. HISP is an exception to this. DHIS 1.x was and still is being developed by a team in South Africa. Most of the contributors to DHIS 2 are from Norway, but there are more contributors from developing countries like Vietnam, India and Ethiopia than what is common in other FLOSS projects. More contributions will most likely come from developing countries when DHIS 2 gets into production.

HISP is also more than a community around a program, it is more importantly a community built around a common interest in improving health services by proper information management. FLOSS is not an end in it self for HISP, it is an important meant to improve health service for the marginalised. However, in the following two section I will concentrate on the motivations to participate in DHIS 2 development and the management of the DHIS 2 development.

Motivation

The ideal developer in a FLOSS project would be a user-programmer who is a competent programmer and the actual user of the system. This devel-

oper would be intrinsically motivated to develop the system, have all the time in the world and demand no pay. To have such developers would be unlikely for any project. Here I will look into how developers can be motivated to participate, and how developers are motivated. I will base this discussion on both of my case studies.

My motives for starting on this thesis and for conducting the two case studies was for once that I wanted to take my degree. This is the cornerstone motivational factor within HISP, as HISP is a research network with master students, PhD students and faculty members. Most of the effort done within HISP is done by researchers doing action research. The downside to this motivation is that it is an extrinsic motivation. Participation in DHIS 2 development is within this frame only a mean to get a degree.

The course held at UiO have the same problem because participation through this course can easily be seen as only a mean to pass the course. Some of the student taking the course can become intrinsically interested in the development of DHIS 2, however. The development of the DHIS 2 core is likely to attract more technically inclined people, or hacker if you will. The DHIS 2 core is developed with a number of interesting frameworks. The frameworks that DHIS 2 builds upon, the tools used to support the development and the development practises used, this are all very handy to have learned for a future work situation. The primary motivation to participate in the development of DHIS 2 is to enhance development skills. The students participating in DHIS 2 development can say they have experience with relevant technologies when applying for a job. People participating in the development of DHIS 2 do not need to be interested in health information per se.

An other important motivation for contributing to DHIS 2 development is work related needs. The translators out in the field needs to fix a bug or needs to make specific changes to the core. In Ethiopia the ICD extension to DHIS 1.3 was made for that reason, and for that reason work is being done in Ethiopia to make the same extension for DHIS 2. Demands met by the translators in different countries, will certainly also in the future make it necessary to make extensions or changes to the core. As DHIS 2 is set into production in more countries and regions this will become a valuable source of feedback and contributions.

My second motivation to write this thesis was because I wanted to look into how FLOSS can benefit developing countries. Like most of the community based FLOSS projects, HISP is not seeking profit. I guess that it is more rewarding to participate voluntarily in making FLOSS software like DHIS better, than to do the same for a company seeking profit. The idealistic foundation of HISP and the value of empowering the marginalised in the

developing world was important for me, it was the reason I chose to write a master thesis withing the HISP network. I am more technically inclined and considered writing a master thesis relating to the *Distributed Multimedia Systems* (DMS) group at UiO, but it felt more valuable to write my thesis within the HISP network.

HISP do pay people to do development and it is not unlikely that some contributions can come from the organisations using the systems when DHIS 2 is set into production. The health authorities in the different countries can heir people to make changes to DHIS 2, or they can give HISP money to heir people. In this instances the paycheck is the primary motivation.

Summing up I think HISP is most likely to motivate people to participate in the development of DHIS 2 through giving students a chance to improve skills, by work-related needs in the field, by idealism and the paycheck.

Management

To harvest contribution to a project it is important how the project is managed. In this section I will look at the eight points in section 5.4 and how this points relates to the management of the DHIS 2 development. I will not restrict the discussion to this points, but will use them as a framework for the discussion.

The first point is to make it interesting and make sure it happens. The project leadership should act as motivators and encourage contributions. It is important that the people who contribute feels that their efforts are appreciated. Patches and suggestions should be seriously considered. The DHIS 2 project is a little bit different from conventional FLOSS projects since not all contributors are strictly voluntary, many contributors takes the INF5750 course where contributions are mandatory and many are master students who have to contribute to get their degree. The students taking INF5750 are not strictly free to what and how much to contribute. The course is important as a base for recruitment of master students and later voluntary contributors. Similar courses can and are being held at other universities in the HISP network.

Closely related to motivation is the concept of “scratching an itch”. Health information is unlikely to be a personal “itch” for a programmer, so contributions based on meeting a need in the developers immediate environment is unlikely. There is other kinds of “itches” that can be “scratched” in the DHIS 2 context. There can be an “itch” to learn to be a better programmer and how to use interesting frameworks. This can help a developer to position himself in a competitive job marked. In other words the “itches” that

can be "scratched" comes from the motivations mentioned in the previous section.

DHIS is designed to avoid having to reinvent the wheel in each country and region DHIS is implemented. The stable core of DHIS contains features relevant across health systems, and the flexible exterior is designed to be relatively easy to change. DHIS 2 also use many FLOSS Java libraries and frameworks to ease the development. In addition to the core framework and development tools, DHIS 2 uses libraries for report generation and data analysis. Sometimes, however, it can make sense to implement some functionality directly into the application without using external libraries. That is if you only need a small fraction of the library's functionality. It can become difficult to manage all the external libraries, each which have its own update life-cycle. It is important to not drown the application in frameworks, this was the primary reason I did not recommend using a pure plug-in framework, at least not what I managed to make, for DHIS 2. The framework added more complexity than it added value, which is exactly what EJB is frequently criticised for.

There have also been some efforts at solving problems through parallel work processes. In connection with DHIS 2 some research and development projects have been conducted apart from the DHIS 2 development. My experimentation with a plug-in framework is one example of this. It make sense to spin off some experimental sub-projects for possible later inclusion in the DHIS 2 code. This sub-project produces more or less throw-away prototypes which can be amended for inclusion in DHIS 2.

To attract contributors and to facilitate the work of the translators, it is important that the entry barrier to effectively use, configure and change DHIS 2 is as low as possible. A sensible interface for configuring the flexible exterior of DHIS 2 is important. For more complex changes and extensions it is important with well documented and understandable code. No matter how good the code is written and no matter how good it is documented, it is not easy to read code. There should be documentation in a more human readable form. There should be documentation for the developers of the DHIS 2 core, for extensions developers, for the translators and for the actual users. A potential DHIS 2 core developer need to understand the architecture of DHIS 2, and the concepts and abstractions used. A potential DHIS 2 core developer also need to understand the frameworks and libraries used. A potential extension developer need to know how to proceed in making an extensions. The translators in the field need to know the interface for configuration, how to submit bug reports and feature requests. The users need to know how to input data, produce reports and how to do data analysis. Documentation for, and perhaps by, these categories of DHIS

2 participants should be made.

Currently DHIS 2 do not benefit from massive peer-review. As DHIS 2 is set into more widespread use I think we will see more bug reports and feature requests from the DHIS 2 translators. This will also create a greater need for support. A lot of support can be given through good documentation, but this is often not enough. To release the core developers from the support burden, there should be mechanisms in place to help the users (the translators and actual users) help each other. This can be facilitated through a user e-mail lists, a support discussion forum, a IRC chat channel or similar means. The advantage of an archived e-mail list or discussion forum is that the support answer get stored, then a user facing similar problems do not have to ask.

DHIS 2 is dependent on other projects. DHIS 2 can function as a proxy for bug reports from its users. Some of the bug reports and feature requests can be related to the libraries and frameworks DHIS 2 depends on. The DHIS 2 developers can make a fix to a bug, or make a patch implementing changes necessary to support a feature request. This can be sent upstream to the relevant library or framework project. The DHIS 2 developers do not need to fix a bug or make a patch themselves. The bug report or feature request can be forwarded to the relevant project. This is similar to the way Linux distributions function as proxies for the Linux kernel and many other pieces of software.

The DHIS 2 project have still not finished a full release of DHIS 2. Up until the time of the full release intermediary milestone releases are made. For each milestone more functionality are implemented. After the full release I think it would make sense to divide the code into two branches. One unstable development branch with cutting edge features, and one stable branch which is debugged and where only features which do not require major code changes and do not break compatibility with extensions, is incorporated. This is similar to the way the Linux kernel is developed. It is a compromise between releasing early and often, and having stable software for production.

11.2.2 Comparing DHIS 1.x and DHIS 2 development

Both DHIS 1.x and DHIS 2 are FLOSS and are conceptually the same application. The major differences between this two different strains of the DHIS software are in how they are developed and on which platform they are implemented. In this section I am going to look at the advantages and

disadvantages of the different development styles and platforms used by this two different strains of DHIS.

The development of the DHIS 1.x strain has since it's beginning predominantly been done by a development team in South Africa. The development style of DHIS 1.x is that of building a "cathedral". It is done by a tightly knit team. The design of DHIS 1.x is most definitely done in a participatory manner, but the development is not. You can say that DHIS 1.x has been developed using participatory design, but not participatory development. The software made in South Africa is transferred to the other nodes in the HISP network, where DHIS 1.x is translated to the local context. The translators give feedback to the development team making the DHIS 1.x core. It is only the core team that really understands the software, which puts a lot of support pressure on this team, and especially its leader Calle Hedberg.

The development of DHIS 2 is built around a community model. The design of DHIS 2 is for the most part based on DHIS 1.4. Years of PD experience is inscribed into DHIS 1.4. The PD approach to design is just as important to DHIS 2. The difference is that by basing the development of DHIS 2 on a community model it becomes participatory development as well as participatory design. Section 2.2.2 discusses the limits of PD. Many of these limits are relevant for any participatory approaches, you need people with sufficient time, skill and motivation. The difference is that our pool of potential participation in DHIS 2 development is not limited to the actual users of the software, and is not limited by any organisational boundary.

In Ethiopia we had to make changes to the stable core as well as to the flexible exterior of DHIS 1.3. In a way this was participatory development, but it was impossible to incorporate the changes into the official releases of DHIS 1.x. This made it impossible for the Ethiopian DHIS 1.3 to benefit from minor releases coming from South Africa. We where stuck with version 1.3.0.17. This was done due to the limitations in the DHIS 1.x architecture and platform. We could have transferred the changes made to this version to new minor releases, but this would place a quite large maintenance burden on the Ethiopian HISP team.

The limitations of MS Access, particularly in the area of modularisation, have given rise to a number of application separated from the DHIS 1.x user interface. This applications are implemented on other platforms, like Java or C++, and have its own user interface and business logic. This applications are linked to DHIS through the use of the database. The reason for making visually separate applications is to avoid making changes to DHIS that makes upgrading to new DHIS versions difficult. In Ethiopia changes to DHIS 1.3 were done through making direct changes to DHIS 1.3, effec-

tively making the Ethiopian DHIS 1.3 a fork. It is extremely difficult to base a distributed development process on MS Access. MS Access is definite not designed with that in mind.

A great advantage with MS Access is that it is quite easy to learn. It is not necessarily more easy to learn for the actual users of DHIS, but it is more easy to learn for the people translating the software to a local context. Both DHIS 1.x and DHIS 2 offers visual configuration for many of the necessary adaptations of DHIS, like the organisational hierarchy and the data elements. There can be times, however, when this is not enough. I did not have any prior knowledge of MS Access before I went to Ethiopia, but I had no greater problems learning enough of MS Access to make necessary changes. It is also more common for developers in developing countries to be familiar with MS Access, than it is to be familiar with Java. It is highly unlikely that a local developer, like Kalkedan in Tigray, would be familiar with Spring, Hibernate, WebWork 2 and the other frameworks used by DHIS 2. I think DHIS 2 will place more demands on the translators if it becomes necessary to make changed not possible through the configuration interface.

MS Access is also great for making prototypes, mostly throwaway prototypes. For DHIS 1.x MS Access also served as a platform for making an evolutionary prototype. This enabled HISP to come up with a working solution relatively fast. Having a working solution is in my opinion the greatest argument any IS project could have. This helped HISP in the competition with other more ambitious projects in South Africa.

MS Access being proprietary software can in principle be seen as a disadvantage. Developers need to have it in order to contribute, the translators need to have it in order to translate the software and the users need to have it in order to use it. I did not have MS Office before I went to Ethiopia and I did not want to buy a license. Fortunately I got MS Office from my institute. MS Office is so ubiquitous in the developing countries where HISP operate, so this is not an important limitation. A more important limitation with the proprietary nature of MS Office is that HISP cannot legally bundle a specific version of MS Office together with DHIS. Therefore HISP have to support many different version of MS Office, and hope the client have one of the supported versions.

The greatest advantage with DHIS 2 is that it is possible to develop it using a community model. In my opinion this is in it self a sufficiently good argument to make a shift from MS Access to Java, or any other open platforms. HISP is after all a distributed community and doing distributed development have the potential of harvesting the collective effort and intellect of the HISP community. Independent of the DHIS 2 software I think it is a

great advantage just having a community web site, which is the case for HISP now. Documentation and support can also be done in a distributed collaborative manner.

In DHIS 2 the "onion architecture" of DHIS 1.x is changed into an object oriented three tire architecture, using Spring to manage the dependencies between the objects. This architecture is more flexible and able to grow and be changed in the future. DHIS 2 has a higher barrier for people unfamiliar with the technology, but it has a lower barrier into understanding the code sufficiently to become a core developer because of a more sensible architecture.

Changing the flexible exterior of DHIS is done through configuration, but sometime it is necessary to change the DHIS core as well. The modularisation of DHIS 2 makes the stable core more flexible without making it fall apart through forking into many incompatible versions. Both the use of OOP and the use of Spring makes DHIS 2 more extensible. It is possible to make local changes and have it integrated with each new minor release of DHIS 2. Currently there are no standard way to make extensions to DHIS 2, however. The use of OOP and Spring makes the applications generally more flexible, but it give no clear route ahead to how a specific extensions, like an ICD extension, should be made. It is possible to extend classes, implement interfaces and change the Spring bean factory to use different classes as dependencies. DHIS 2 still lacks a general architecture for extensions, however.

DHIS 2 also offers an API which other applications can use. In DHIS 1.x other applications could only interface with DHIS through the data base, in DHIS 2 other applications can interface with the business layer. This applications will still be visually separate from DHIS 2 and are therefore not extensions, but more like other applications talking to DHIS 2. Using an API will still be handy for possible future extensions, since an API is less likely to change than the implementation.

In conclusion to the question of extensibility of DHIS 2 I will say that it is still possible to make the core of DHIS 2 more extensible. Using a pure plug-in framework is not the route to go, but it is possible to use less invasive methods, which do not require embedding a java servlet container in the plug-in framework. Using hooks is one route to go. The ICD module which the Ethiopian HISP node is making is excellent for testing the extensibility of the current DHIS 2. Can this be done without creating a dependency on a specific minor version of DHIS 2? Different ways to achieve extensibility should be experimented with in the future, and documentation on how to proceed making an extension should be made.

11.2.3 Comparing the Tigray and DHIS 2 cases

The Tigray case and the DHIS 2 case are very different primarily because I performed different roles in fairly different settings. In both settings I did some programming, but at very different stages in the development. In this section I am going to compare the role I played in Tigray with the role I played in the DHIS 2 development. I am also going to compare the different context in which I operated in the two cases.

In Tigray I worked as a DHIS translator, translating DHIS to the context of the Tigray health system. In this case the social aspects of IS was the most important and took the most time. First we had to negotiate with the Tigray health bureau to set up a client-system infrastructure. This involved a lot of social brokering. In this setting the DHIS software was secondary to the general goal of improving information management. The most important artifact we could offer Tigray was DHIS, but we could also offer help in developing an EDS based on an action oriented approach. I personally could not offer much help in developing an EDS, but the other team members had an more extensive information management background than I had.

For various reasons which are better explained by (Damitew and Gebreyesus 2005), there was a large number of data elements being reported in Tigray. Different departments demanded different reports, and the data elements often overlapped between the reports. It is my opinion that the work we did in Tigray can be justified solely by getting the different departments to work on an EDS. By bringing attention to the data being gathered, redundancy in reporting can hopefully be reduced, and less and more useful data can be gathered. This is an example of a situation where less is more. As we saw in section 8.4 the team appointed to develop an EDS did not actually come up with a *minimal* data set, but it was a reduction compared to the number of data elements that was currently in use.

The technical aspects was also important. We had to configure DHIS to be useful in the Tigray context and import some data from EPI-info to show that DHIS was a viable system. If we did not have the DHIS software we would not have much to put on the table in our negotiation with the bureau. I dedicated my time predominantly to configuring DHIS. Through this work I became familiar with the strengths and weaknesses of DHIS 1.3. In Tigray we was essentially cut of from the rest of the HISP community because of the slow and unreliable Internet connection. In this case a supportive virtual community would not help much, we could, after all, not access a virtual HISP community. I had downloaded a lot of documentation for Python allowing me to make the EPI-info to DHIS transfer script. Documentation for MS Access and VBA was also on my hard disk. To en-

able DHIS translators to work in places with no useful Internet connection, documentation for DHIS 2 should also be available in a form that can be downloaded to the local hard drive.

During my stay in Addis Ababa I had an usable Internet connection, and here I missed having contact with a virtual community accessible by other means than e-mail. I had no idea on who to contact about my GIS questions, and sending e-mails back and forth takes time. For the translators out in the field it would be helpful to have a responsive virtual community to help in the configuration of DHIS, and perhaps also to discuss various dilemmas of a more social nature. There are people in the HISP community who have a lot of experience from negotiating agreements with health authorities. There are also people with experience from conducting DHIS training sessions. In Tigray I had to make all the training material myself. I missed having access to training material. Training sessions has been conducted in every country in the HISP network. There must be a lot of material already made[1]. The context for training varies but DHIS is the same, so training material made in other context can still be useful.

In the DHIS 2 case I worked as a developer of the DHIS 2 core, not on the core itself but on an experimental sub-project. This role challenged my technical skills more than my social skills. The challenges I met in the Tigray case was predominantly social in nature. I was a stranger in a foreign county. I did not know the local language or culture, and I was not used to the local bacterias. In the DHIS 2 case the challenges was technical. I started out with a hope to make a plug-in framework for DHIS 2. This turned out to be a serious technical challenge.

The DHIS 2 case I conducted in Norway. A much better Internet connection and familiar surroundings made it much easier to work with software development. When I started on the DHIS 2 case a community web site was in the process of being made. This web site gave HISP a more clear point of presence on the Internet, and in a way this has made HISP more "real". What really makes HISP real is the people involved in the process of structuration of HISP, but the web site makes the sense of structure more clear. Over the years there has been made many small applications in the process of translating DHIS 1.x to a new context. Most likely an ecosystem of related applications will be built around DHIS 2 as well. It would make sense for the HISP web site to offer hosting services for this applications as well.

In the DHIS 2 case I did not have any contact with any translators or actual users. The DHIS 2 development was in its infancy and the core design

[1] At the South African HISP web site (http://www.hisp.org/) training material for DHIS 1.x is now available

was taken from DHIS 1.4. Unfortunately the plug-in framework project I worked with became isolated. The technical challenges became too much for me to handle within my time constraints. In retrospect I realise I had a too grand vision for the plug-in framework. There did not exist any plug-in framework that could meet our vision of a pure plug-in framework for web applications. We made a throwaway prototype as a proof of concept, but making DHIS 2 into a pure plug-in application would add a lot of complexity to the coding of DHIS 2. Making a pure plug-in application involves a way of thinking unfamiliar to most programmers. I realise that I should have involved the other modules more into a discussion about how to make DHIS 2 extensible. The technical challenges I meet overshadowed the need to discuss how to provide extension points and where in the other modules this extension points should be. Extensibility is something that involves all the modules of DHIS 2. I should have focused less on making a plug-in framework and more on discussing with the other groups on making their modules extensible with the frameworks already in use.

DHIS 2 is currently being tested in India, Vietnam and Ethiopia. It will be interesting to see the experiences the translators in this counties have with configuring and extending DHIS 2. MS Access is well known among moderately competent developers in developing countries. In Ethiopia we hired Kalkedan to maintain and support DHIS for one year at the Tigray health bureau. He had tree years education from Mekelle University, where he had learned Visual C++, VB and MS Office. He had no knowledge of Java, and certainly not of Spring, Hibernate and WebWork. So the question I ask myself is: Would Kalkedan be able to configure and extend DHIS 2? I think he would be able to configure DHIS 2. Using an interface for visual configuration is not that difficult. If he, however, had to change or extend the code I think he would have a hard time. For DHIS 2, I think it is important to have an active virtual community where translators and the organisations using DHIS 2 can get support. It is important that the organisations where DHIS 2 are used are able to effectively use DHIS 2 without direct help from outsiders, except the help provided through the virtual community. The organisations using DHIS 2 should not perceive themselves as consumers of DHIS 2, but as collaborators in making a great application for information management in the primary health system.

In the Tigray case I worked as an DHIS translators. In the DHIS 2 case I worked as a DHIS 2 core developer. It has been a tremendous challenge to perform both these roles. I had to learn social science theory to perform my work as an DHIS translator and in order to write about the Tigray case. In order to perform my role as a core programmer I had to have programming skills, and I had to learn a lot of new technologies. It would be advisable to let new master student perform just on of these roles. The technical inclined

can work with the DHIS core and the social inclined can work as DHIS translators. Dependent on the problem definition of the thesis the master student plans to write one or both of these roles can be performed.

11.3 Theoretical considerations

In this section I am going to make some theoretical reflections based on conventional IS theory and FLOSS. First I will use ST to reflect on the formation and social practices of the FLOSS community. Then I will reflect on PD, HISP and FLOSS and lastly I will make some reflection about technology translation.

11.3.1 Structuration of FLOSS

In chapter 4 I gave a historical narrative to show how the social practices of FLOSS have changed and replicated over time and space. The practice of code sharing is not a newly invented practice, but has been present since the dawn of computing. As long as there was no business interest in selling software the natural thing to do was to share code. The only reason to keep code secret is if it enable you to sell software, or if it gives you a competitive advantage to keep a piece of code secret.

The social practice of sharing code is the most important social practice in the FLOSS community. Individual actors have different motivations to do this. Some do it for moral reasons because they have the opinion that code should be free. Others do it because of more pragmatic reasons. It is my opinion that the true test of whether source code is shared for moral reasons, is if there is a business opportunity in keeping the code secret, but where the developer chooses to share the code anyway.

The practice of code sharing based on pragmatic reasons have emerged in places where there are no interest or no opportunity in selling software, and where the network effect of sharing code is greater than the competitive advantage it gives to keep the code secret. In the case of Unix, AT&T had no opportunity to sell software (not until the 1980s anyway). The academic computer science research community quickly adopted Unix as an experimental toy. This community got its money from doing research, not from selling software. Sharing information is a basic value in the academic research community that enable researcher to “stand on the shoulders of giants”. The academic community do not always practice this value, often pride and glory seeking gets in the way. The Unix users did not share code

for any moral reason, but because the network effect of sharing was that you got access to much more software than you provided.

An other important lesson to draw from the Unix history is that it is more important to make something work, than it is to implement a grand vision. The most important reason Multics did not succeed was that it sought to implement a grand vision, to complex for the state of technology at the time. Unix became a success not because it was a technical master peace, but because it worked and was available. Most important, the source was available. When you have something that work you can attract the attention of people willing to contribute if you are willing to share. This was what Unix did. Even if the first instant of the software is simple, a more complex and modular design will emerge if it attract contributions from more than a few programmers. The modular design will not emerge spontaneously, but to manage the contributions and to keep the interest up it is forced into existence.

Linus Torvalds did not make Linux because he was interested in selling it, but because he wanted an operating system he could experiment with. The first version of Linux was to simple to sell anyway. Linux would not have been possible without many people contributing. The more morally motivated FSF worked on the Hurd kernel. The Hurd kernel did not succeed because it did not deliver, being based on a too ambitious micro kernel design. The Minix kernel was available, but Tanenbaum was not interested in making it into anything more than a learning tool. Tanenbaum did not step up to manage all the contributions. Linux was available and Torvalds did step up to the challenge. Linux have, from its humble beginning, evolved into a kernel with a modular design facilitating contributions.

In the TCP/IP and OSI-stack case we see this story again. One grand vision, the OSI standard, failed because it was too complicated. Even if it was technically superior to TCP/IP it was not a working solution. TCP/IP was working and had been working for some years. Over the years the TCP/IP protocol stack have evolved and other protocols have been made at the application layer to implement some of the functionality present in the OSI standard. On the other hand, if TCP/IP was too simplistic it would not have attracted as large a following. TCP/IP did replace the more simplistic UUCP protocol. UUCP is much more cumbersome to use compared to TCP/IP because of its simplicity.

My opinion is that the Unix and Linux stories illustrates the importance of modular design and communicative code in distributed development common in FLOSS projects. Modular design and distributed development is not limited to FLOSS project, but in FLOSS project of some size it is a necessity. To build a modular application with easily understandable code

require more work up front, but in the long run it is a great assess. The cost of maintaining software is usually much higher than the cost of first making the software, and it is easier to maintain good quality code. The Unix, Linux and Internet protocol stories also show that it is more important to have a simple working solution than to have a grand vision. There is a conflict of interest between modular design and to have a simple working solution. Modular design require more work up front. By designing a system with too much extensibility you risk never being able to deliver. This was the case for the Hurd kernel and for my plug-in framework. A balance between a simple working solution and modular design has to be struck.

In addition to the practice of sharing code there are other important social practices in the FLOSS community. The social practise of structuring a project around a piece of software is one example. It is possible to just give software to others without starting a project. The practice of listing important contributors in a credits file is another example. Many of these social practices are mentioned in chapter 5. In Figure 11.2 I have made a model of the modalities of signification, domination and legitimisation giving structure to the FLOSS community. The figure also show other social systems influencing FLOSS. The line are intentionally dashed to show that the boarder between the social systems are open ended.

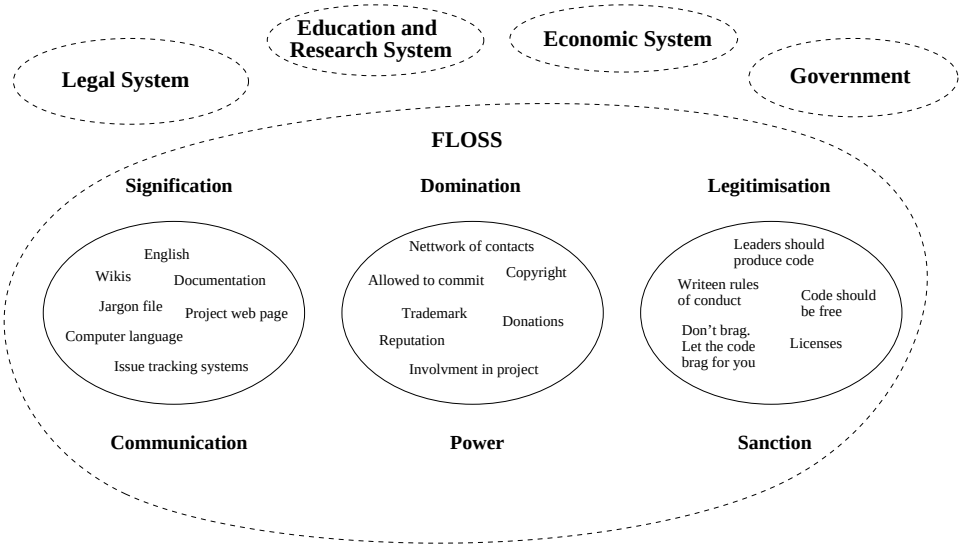

Figure 11.2: FLOSS modalities

11.3.2 Participatory development

Within the HISP network the participatory design strategy is important for the development of DHIS and even more for the translation of DHIS to a new context. It is important to seek user feedback throughout the design process. In Tigray we involved the users through the formation of a team to decide on data elements and reports. Most of the reports that this team decided on could not be used in a computer based IS. Regular users often need guidance throughout the design process to be able to participate effectively. Users often lacks a clear understanding in what is technical feasible. For this reasons I made significant changes to the reports decided by the user team, and even added and removed some data elements. I sought user feedback by asking the departments what they wanted the reports to contain, and by showing them the reports I had made.

PD focuses solely on the design of a system. If the design is isolated from the actual implementation of the software, there is good chance that the design obtained through user involvement becomes bloated and difficult to implement. User feedback should not only be sought throughout an isolated design phase, but should be sought after throughout the actual implementation of the system as well. If reality shows that some design decisions cannot be met, user feedback on the changed design should be sought.

On the FLOSS scene it is not uncommon for some of the actual users of a piece of software to be programmers as well as users. In these cases it makes sense to extend the term participatory design into participatory development. It is not necessary to seek user feedback for project where the actual users are also programmers, feedback comes to the project in the forms of patches, bug reports and feature requests. For projects where the actual users are not programmers it is unreasonable to expect users to be able to contribute patches or to give precise bug reports. Regular users can contribute with feature requests and give superficial bug reports, however. Regular users can also contribute with user support and user documentation. Feedback from regular users have to be actively sought after, and they generally need more guidance in communicating feature requests and bug reports.

To assure continuous user feedback it makes sense to develop an evolutionary prototype. It is much easier to comment on an actual running system compared to mock-ups. All FLOSS software are in a sense evolutionary prototypes, constantly changed at the same time as it is in active use. FLOSS takes small steps at the time, no grand releases after an extended period of isolated development. Throwaway prototypes can be well suited for

experimenting with different solution to a difficult problem. Experimentation with different solution can be done in parallel and the most suited solution can be incorporated in the evolutionarily prototype. Throwaway prototypes are also useful for getting user feedback in the initial design phase.

11.3.3 Free to translate technology

Relating to my presentation of technology translation in section 2.3.2. FLOSS offers unique opportunities for translating software, both translation of the language and the functionally, making software more useful for a developing country. It also takes the term *configurable* to a new level. FLOSS give the control of the software to the user, and configuration of software can be done down to the source code level. Source code changes requires advanced computer knowledge and hard work, this is not an option for most individual users and small non-computer related businesses. For a country, however, this is indeed possible.

Three points was mentioned in connected with the presentation of technology translation:

1. The initiative should be designed as an incremental and context sensitive process, carried out in rather small steps.
2. Translation represents and iterative end evolving long term process, having implications for both sustenance and scale issues.
3. Technology translation includes building and supporting heterogeneous socio-technical networks and ensuring indigenous capacity building.

This points are relevant whether the software is FLOSS or not, but FLOSS facilitate indigenous capacity building. It is in the process of customisation that the availability of the source code is especially relevant. See Figure 2.5.

DHIS is built to enable visual customisation. It is common in proprietary software as well as FLOSS software to enable customisation. If it was possible to do all the necessary changes to DHIS in the translation process, through customisations, then DHIS could be distributed with a non-FLOSS license. This has not been the case, however. Frequently there has been situations where source code changes were necessary. For context insensitive software like operating systems and word processors the customisations

offered by the software is usually enough. For more context sensitive applications like DHIS the customisations offered is often not enough. In this cases it is a life saver to have access to the source code. The ability to change source code empower the people translating software, both those who do language translation and those who change the design and functionality of an application.

Chapter 12

Conclusion

In this chapter I will first give an assessment of the validity of my research according the validity criteria described in section 3.1.1. Then I will make some concluding remarks about my research into FLOSS, HISP, Ethiopia and developing countries. Last I will look into the future and give some suggestions for research into my problem domain.

12.1 Validity of the research

My research was conducted using AR. Within AR the research situation determine which concrete research methods to be used. This can be both qualitative and quantitative methods. AR do not have any absolute criteria to determine if the interpretations made are valid. AR research will always be open to new interpretations. To give some idea of how valid my research is I will use the validity criteria mentioned in section 3.1.1.

12.1.1 Process validity

To reach appropriate conclusions in AR it is important for the research to go through many cycle. At the end of each cycle intermediate conclusions are made. Based on this intermediate conclusions better research questions can be asked. Through my research I have constantly refined my understanding of FLOSS, HISP, Ethiopia and developing countries through direct and vicarious experiences. I have read a lot of literature, I have participated in FLOSS development in Ethiopia and in the DHIS 2 project. In my research I have gone through many small cycles. As Figure 3.2 goes I have only gone

through one full circle for each of my two case studies. This thesis is the first real result of the *Specifying Learning* phase.

Throughout my research I have made use of triangulation. I have made use of both qualitative and quantitative methods to verify my conclusions. I have also used many information sources like my own experiences, the experiences of my coworkers and stakeholders, and various literature. All in all I have been quite rigorous in my information handling. I developed a good relationship with the other members of the Tigray team, the relationship developed with the workers in the Tigray health bureau and in the two district was limited. During the one week training at the health bureau I got a quite good relationship with the workers. They had no problems with asking questions and coming with feature request. In the DHIS 2 case I unfortunate developed a too limited relationship with the others in the DHIS 2 virtual community.

12.1.2 Dialogic validity

Within AR it is considered important to subjugate research data, methods and interpretations to peer review. In the process of writing this master thesis I have only done this to a limited extent. I made one interview with Knut Staring, which is one of the managers of the DHIS 2 project, to obtain data about DHIS 2 and to verify my assumptions of the DHIS 2 project. In Tigray I naturally talked with the other team members about what we should do, and about what we experienced in our dealings with the health bureau. Hopefully other people will read this thesis and be able to improve my interpretations.

12.1.3 Outcome validity

The outcome of the Tigray case was not exactly what we hoped for. DHIS 1.3 never came into regular use, not even in the pilot districts. We succeeded, however, in bringing much needed attention to the data gathering practices in Tigray. Tigray have expressed interest in DHIS 2, and perhaps this system will come into regular use in Tigray. This still remains to be seen. It was not a goal in it self for Tigray to use DHIS 1.3, the goal was to improve the information handling practises, and we assisted the Tigray health bureau in this. It was their choice to not start using DHIS 1.3 in the pilot districts.

When I started on the DHIS 2 case I was hoping to build an ICD code plug-in for DHIS 2. From the experiences I got in Ethiopia I saw the need for

such a plug-in to DHIS 2. Unfortunately I never came around to actually build this plug-in. The DHIS 2 project was at an early stage of development and it was not clear how DHIS 2 should facilitate extensibility. I worked on a plug-in framework for DHIS 2, and I managed to make a throwaway prototype of a framework. I concluded that this framework would create more complexity than it would give value in extensibility.

12.1.4 Catalytic validity

I have most definitely moved towards a better understanding of FLOSS, HISP, Ethiopia and developing countries. I have also moved towards a better understanding of IS theory, and I have improved as a programmer. The knowledge and experiences I have obtained through this research will be of great assistance in a future work situation. The DHIS 2 case have taught me important lessons about software development and I have learned to use interesting technologies. Using AR to do practitioner study on my own work situation as I did in the DHIS 2 case will also make me more self-reflective in a future work situation. My knowledge about FLOSS will make it easier for me to participate in FLOSS projects, and my experiences from Ethiopia have made it easier for me to work with people from other cultures. All in all I have grown as a human being, and I feel prepared to work with programming in various social contexts.

It is more difficult to assess how much the workers in the Tigray health bureau have moved to a better understanding of their organisation and information handling practices. Damitew and Gebreyesus (2005) can tell more about this than I. I do not think I have contributed much to make the other DHIS 2 developers understand FLOSS and the DHIS 2 project better, at least not while I conducted my research. I do not think anybody but myself have moved significantly to a better understanding of FLOSS, neither in the Tigray case or in the DHIS 2 case, through the research process. However, those who choose to read this thesis can learn a lot about FLOSS, HISP, IS theory, Ethiopia and developing countries.

12.1.5 Democratic validity

In the Tigray case the research was most definitely done in collaboration with the stakeholders. We offered our services to the Tigray health bureau and they accepted. In the Tigray case we acted as facilitators, facilitating discussions about the data gathering practises in Tigray. We also acted as software developers, configuring DHIS 1.3 for Tigray. User feedback was

sought through the training sessions, and by questioning the different departments about the reports DHIS produced or should produce. The DHIS 2 case was a practitioner study and I should have sought different views on how to make DHIS 2 extensible from the other DHIS 2 developers. Both our research in Tigray and my research in the DHIS 2 project was relevant to the local context.

12.2 Concluding remarks

This thesis have explored FLOSS, HISP, Ethiopia and developing countries. I have explored how FLOSS are and can benefit HISP, and I have explored how FLOSS are benefiting and can benefit Ethiopia in particular, and developing countries in general. This thesis has given a general introduction to FLOSS and HISP. I have presented one case study conducted in Ethiopia, where I worked as a DHIS translator. This case study was used to investigate FLOSS and Ethiopia, and HISP and the Tigray health bureau. I conducted another case study as part of the DHIS 2 project. This case study was used to explore FLOSS and HISP.

This are exiting times for FLOSS. The practice of source sharing and distributed voluntary development has risen from obscurity in the last few years. FLOSS offers an opportunity for technological marginalised countries to benefit from a lot of freely available information, both in source code form and in English (and to a lesser extend other languages). The source code opens up the black box of software in ICTs, and gives developing countries the opportunity to translate the software into the local context.

In the Ethiopian region of Tigray we used the freedom offered by FLOSS to translate DHIS 1.3 into the local context of Tigray. We would be unable to perform our job in Tigray if we did not have access to the source code. In Tigray the social aspects of IS development was the most important. The challenges we met was both technical and social in nature, but the social aspects was most important. Most of the time was spent on negotiating with the health bureau, conducting training and trying to get the different departments to talk together. As the only foreigner on the team I meet additional cultural and language challenges. I concentrated most on the technical aspects of configuring DHIS 1.3 to the local context of Tigray.

The awareness of FLOSS is low in Ethiopia compared to more technically advanced developing countries. For the few individuals in Ethiopia who have their own PC there are no clear advantage in using FLOSS. Cheap pirated versions of proprietary software are readily available. For the education sector and the general government sector, like the health system,

FLOSS has a more clear advantage. Ethiopia have signed the TRIPS agreement, which require Ethiopia to ratify intellectual property right laws. This give the Ethiopian government an economical incentive to use FLOSS. Independence from software vendors by promoting open standards, less vulnerability to viruses and the possibility of fostering home grown computer industries are more important than the direct monetary savings. For Ethiopia to benefit from and contribute to FLOSS local capacity have to be fostered through the education sector. Some minor institutional support for FLOSS are in the process of being built in Ethiopia through efossnet.org.

The DHIS software has since its inception been licensed with a FLOSS license. However, the software was not developed using distributed voluntary development. The development was done by one team in South Africa. Due to technical limitations of the platform DHIS was developed on, distributed development was difficult. To overcome the limitations inherent to the platform DHIS was developed on, it was decided to start the DHIS 2 project. DHIS 2 is a total reimplementation of DHIS based on Java and Java frameworks. The single most important advantage of DHIS 2, is the possibility it offer for doing distributed voluntary development. Through participating in the DHIS 2 project I learned more about participating in a FLOSS project, and I learned more about HISP. To facilitate distributed development it is important with modular well documented code, good development documentation and an active virtual community. To facilitate the creation of modules which can be distributed independent of the DHIS core it is important with extensibility. Not all modules of DHIS are equally relevant in all contexts. How extensible DHIS 2 is and will become remains to be seen.

This thesis has been conducted within a social informatics perspective which recognise the importance of the social and technical context in which an IS is applied. Technology cannot be seen to have a predetermined effect on the social system in which the technology is applied. The social system is often changed by the technology in an unanticipated way, and the technology itself is also changed by the social system. Important to the understanding of how social system are changed and replicated over time and space is the grand theory of structuring theory.

12.3 Possible future research

My research into FLOSS and Ethiopia is rather broad. To find ways to effectively benefit from FLOSS there is a need for more focused research. I have identified the education and research sector of Ethiopia as a sector where

Ethiopia is most likely to benefit from FLOSS. Research into how this can be done most effectively would be of interest. More general research on the usage of FLOSS in the education and research sector in both developed and developing countries would also be of interest.

As seen from Figure 1.1 I have not focused on the use of FLOSS in health care and FLOSS health applications, and neither have I investigated to any detail how the Tigray health system relates to the overall Ethiopian health system. This are research area I leave to others. This thesis has not focused on health care. DHIS is one health care application, but there exist others that could be of interest to both HISP and Ethiopia.

Unfortunately I never came around to make DHIS 2 extensible. Research into how to extend DHIS 2 and how to make DHIS 2 more extensible would be an interesting research questions, for the more technical inclined.

References

Asaro, Peter M. (2000). Transforming society by transforming technology: the science and politics of participatory design. *Accounting Managment and Information Technologies* (10), 257–290.

Baark, Erik and R. Heeks (1998). Evaluation of donor-funded information technology transfer projects in china: A lifecycle approach. (http://www.sed.manchester.ac.uk/idpm/publications/wp/di/). *Development Informatics: Working Paper Series*.

Baskerville, Richard L. (1999). Investigating information systems with action research. (http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.html). *Communication of the Association for Information Systems 2*.

Berhe, Aregawi (2004). The origins of the tigray people's liberation front. *African Affairs 103*(413), 569–592.

Bjerknes, Gro and T. Bratteteig (1995). User participation and democracy: A discussion of scandinavian research on systems development. *Scandinavian Journal of Information Systems 7*, 73–98.

Braa, Jørn (1997). *Use and Design of Information Technology in Third World Contexts with a Focus on the Health sector: Case studies from Mongolia and South Africa*. Ph. D. thesis, University of Oslo.

Braa, Jørn and B. Blobel (2003). Strategies for developing health information systems in developing countries. Technical report, WITFOR Health Commission.

Braa, Jørn, E. Monteiro and S. Sahay (2004, September). Networks of action: Sustainable health information systems across developing countries. *MIS Quarterly 28*(3).

Briggs, Philip (2003, February). *Ethiopia - The Bradt Travel Guide* (3rd ed.). Bradt Travel Guides Ltd.

Bødker, Susanne and K. Grønbæk (1991). Cooperative prototyping: Users and designers in mutual activity. *International Journal of Man-Machine Studies 34*(3), 453–478.

CDCP (2005). About epi info. WWW. (http://www.cdc.gov/epiinfo/about.htm).

Crinnion, John (1991). *Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology*. Plenum Press, New York.

Cross, Michael (2005, August). Ethiopia's digital dream. (http://technology.guardian.co.uk/online/story/0,3605,1541785,00.html). *The Guardian*.

Damitew, Hirut Hebrekidan and N. H. Gebreyesus (2005). Sustainability and optimal use of health information systems. Master's thesis, University of Oslo.

Davis, Alan M. (1992). Operational prototyping: A new development approach. *IEEE Software 9*(5), 70–78.

Denzin, Norman K. and Y. S. Lincoln (Eds.) (1994). *Handbook of Qualitative Research* (1 ed.). Sage Publications, Inc.

Dick, Bob (1993). You want to do an action research thesis? www. (http://www.scu.edu.au/schools/gcm/ar/art/arthesis.html).

(Director), Samia Zekaria (2005). Ethiopia demographic and health survey. Technical report, Ethiopia Central Statistical Agency.

Docktor, Roslyn. Accelerating e-government ...e-readiness at work. Technical report, McConnell International. (http://unpan1.un.org/intradoc/groups/public/documents/CAFRAD/UNPAN006617.pdf). Presenation slides.

Giddens, Anthony (1984). *The Constitution of Society: Outline of the Theory of Structuration.* Cambridge: Polity Press.

Greenwood, Davydd J. and M. Levin (1998). *Introduction to Action Research*. Sage Publication.

Gregor, Shirley (2005). The struggle towards an understanding of theory in information systems. In *Information Systems Foundation: Constructing and criticising*.

Gurstein, Michael (2003). Effective use: A community informatics strategy beyond the digital divide. (http://www.firstmonday.org/issues/issue8_12/gurstein/). *First Monday 8*(12).

Hanseth, Ole and E. Monteiro (1998, August). Understanding information infrastructure. www. (http://heim.ifi.uio.no/~oleha/Publications/bok.pdf). Manuscript.

Heeks, Richard (1999). The tyranny of participation in information systems: Learning from development projects.

(http://www.sed.manchester.ac.uk/idpm/publications/wp/di/). *Development Informatics: Working Paper Series*.

Heeks, Richard (2002). Failure, success and improvisation of information systems projects in developing countries. (http://www.sed.manchester.ac.uk/idpm/publications/wp/di/). *Development Informatics: Working Paper Series*.

Heeks, Richard, D. Mundy and A. Salazar (1999). Why health care information systems succeed or fail. (http://www.sed.manchester.ac.uk/idpm/publications/wp/igov/). *Information Systems for Public Sector Management: Working Paper Series*.

Herr, Kathryn and G. L. Anderson (2005). *The Action Research Dissertation: A Guide for Students and Faculty*. Sage Publication.

Horstmann, Jutta (2004, April). Looking for penguins at the horn of africa. Technical report, Relevantive AB. (http://www.relevantive.de/et_travelreport.html). Travel report.

Kimaro, Honest C. and O. H. Titlestad (2005). Challenges of user participation in the design of a computer based system: the possibility of participatory customisation in low income countries. International Federation for Information Processing: Work Group 9.4. (http://www.hisp.info/confluence/display/HISP/ResearchPapers).

Kling, Robert, H. Crawford, H. Rosenbaum, S. Sawyer and S. Weisband (2000). Learning from social informatics: Information and communication technologies in human contexts. Technical report, Center for Social Informatics. (http://rkcsi.indiana.edu/).

Kohn, Alfie (1987, January). Studies find reward often no motivator. www. (http://www.gnu.org/philosophy/motivation.html).

Lakhain, Karim R., B. Wolf and J. Bates (2002, January). *Hacker Survey*. The Boston Consult Group. (http://www.bcg.com/opensource/BCGHACKERSURVEY.pdf). Downloaded July 15 2005.

Lancasshire, David (2001). Code, culture and cash: The fading altruism of open source development. (http://www.firstmonday.org/issues/issue6_12/lancashire/). *First Monday 6*(12).

Lee, Fion S. L., D. Vogel and M. Limayem (2002). Virtual community informatics: What we know and what we need to know. Hawaii International Conference on Systems Sciences. (http://csdl.computer.org/comp/proceedings/hicss/2002/1435/08/14350214b.pdf).

Mockus, Audris, R. T. Fielding and J. D. Herbsleb (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology 11*(3).

Muehlig, Jan and J. Horstmann (2004, February). Open source project ethiopia. Technical report, Relevantive AG. (http://www.relevantive.de/et_project.pdf).

Nhampossa, José Leopoldo (2006). *Re-Thinking Technology Transfer as Technology Translation*. Ph. D. thesis, University of Oslo.

Perens, Bruce (1999, February). It's time to talk about free software again. (http://lists.debian.org/debian-devel/1999/02/msg01641.html). Archived e-mail.

Proulx, Serge and G. Latzko-Toth (2005). Mapping the virtual in social sciences: On the category of "virtual communities". (http://www.ci-journal.net/viewarticle.php?id=80). *The Journal of Community Informatics 2*(1), 42–52.

Rajani, Niranjan (2003). *Free as in Education*. OneWorld Finland and KEPA Helsinki, Finland.

Raymond, Eric S. Jargon file. www. (http://www.catb.org/jargon/).

Raymond, Eric S. (2001, February). *The Cathedral and the Bazaar*. O'Reilly. (http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/).

Robert K. Yin, Donald T. Campell (2003). *Case Study Resarch: Design and Methods*. Sage Publications.

Rose, Jeremy (2000). *Information systems development as action research - soft systems methodology and structuration theory*. Ph. D. thesis, Aalborg University. (http://www.cs.auc.dk/~jeremy/pdf%20files/thesis.pdf).

Rose, Jeremy (2001). Evaluating the contribution of structuration theory to the information systems discipline. www. (http://www.cs.aau.dk/~jeremy/pdf%20files/ECIS1998.pdf).

Rose, Jeremy and R. Scheepers (2001, June). Structuration theory and information systems development - framework for practice. pp. 217–231. Global Co-Operation in the New Millenium: The 9th European Conference on Information Systems. (http://csrc.lse.ac.uk/asp/aspecis/20010096.pdf).

Rudd, Jim, K. R. Stern and S. Isensee (1996, January). Low vs. high-fidelity prototyping debate. *interActions*, 76–85.

Sawyer, Steven and H. Rosenbaum (2000). Social informatics in the information sciences: Current activities and emerging directions. *Informing Science 3*.

Software Productivity Consortium (1997, June). *Evolutionary Rapid Development*. Software Productivity Consortium. SPC document SPC-97057-CMC.

Spinuzzi, Clay (2002). A scandinavian challenge, a us response: methodological assumptions in scandinavian and us prototyping approaches. pp. 208 – 215. ACM Special Interest Group for Design of Communications.

Stallman, Richard (1999, January). The gnu operating system and the free software movement. In *Open Sources : Voices from the Open Source Revolution* (1st ed.). O'Reilly & Associates, Inc.. (http://www.gnu.org/gnu/thegnuproject.html).

Stallman, Richard M. (2002, October). Why software should not have owners. GNU Press. (http://www.gnu.org/philosophy/why-free.html).

Walsham, Geoffrey (1993). *Interpreting Information Systems in Organizations*. John Wiley & Sons Ltd.

Walsham, Geoffrey (2001). *Making a World of Difference: IT in a Global Context*. John Wiley & Sons Ltd.

Warschauer, Mark (2002). Reconceptualizing the digital divide. (http://www.firstmonday.org/issues/issue7_7/warschauer/). *First Monday 7*(7).

Weber, Steven (2004). *The Success of Open Source*. Harvard University Press.

Williams, Robin and D. Edge (1996). The social shaping of technology. *Research Policy 25*, 856–899.

Williams, Sam (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly & Associates, Inc.. (http://www.faifzilla.org/). Online version.

World Trade Organisation (1994, April). *Trade-Related Aspects of Intellectual Property Rights*. World Trade Organisation. (http://www.wto.org/english/docs_e/legal_e/legal_e.htm). Annex 1C of the agreement.

Yamagata, Hiroo (1997). The pragmatist of free software. WWW. (http://hotwired.goo.ne.jp/matrix/9709/5_linus.html).

Part VI

Appendixes

Appendix A

Lists

A.1 List of Acronyms

AAU ..... Addis Ababa University
ADSL ..... Asymmetrical Digital Subsciber Line
AMD ..... Advanced Micro Devices
ANC ..... African National Congress
API ..... Application Programming Interface
AR ..... Action Research
BIND ..... Berkley Interned Naming Daemon
BSDI ..... Berkley Software Design Incorporated
BSD ..... Berkley Software Distribution
BTL ..... Bell Telephone Laboratories
CARD ..... Collaborative Analysis of Requirements and Design
CDC ..... Centers for Disease Control and Prevention
CITCC China International Telecommunication Construction Corporation
CI ..... Community Informatics
CMS ..... Content Management System
CMU ..... Carnegie-Mellon University
COM ..... Component Object Model
CUD ..... Coalition for Unity and Democracy
DARPA ..... US Defence Advanced Research Project Agency
DBMS ..... Database Management System
DECUS ..... DEC User Group
DHIS ..... District Health Information Software
DMS ..... Distributed Multimedia Systems
DPC ..... Decease Prevention and Control department
EASSy ..... East African Submarine Cable System
EDS ..... Essential Data Set
EJB ..... Enterprise Java Beans
EPLF ..... Eritrean People's Liberation Front

EPRDF . . . . . Ethiopian Peoples' Revolutionary Democratic Front
ETC . . . . . Ethiopian Telecommunications Corporation
FLOSS . . . . . Free/Libre/Open Source Software
FOSSFA . . . . . Free and Open Source Software Foundation for Africa
FSF . . . . . Free Software Foundation
FWA . . . . . Fixed Wireless Access
GCC . . . . . GNU Compiler Collection
GDB . . . . . GNU Debugger
GE . . . . . General Electrics
GFF . . . . . Ge'ez Frontier Foundation
GIS . . . . . Geographical Information System
GPL . . . . . GNU General Public License
GUI . . . . . Graphical User Interface
HDI . . . . . Human Development Index
HISP-ET . . . . . Health Information System Program in Ethiopia
HISP . . . . . Health Information Systems Programme
HIS . . . . . Health Information System
HMIS . . . . . Health Management Information Systems
IANA . . . . . Internet Assigned Number Authority
ICD . . . . . The International Classification of Deceases
ICT . . . . . Information and Communication Technology
II . . . . . Information Infrastructure
IM . . . . . Instant Messaging
INF5750 . . . . . Open Source Software development
IoC . . . . . Inversion of Control
IPC . . . . . inter process communication
IRC . . . . . Internet Relay Chat
ISO . . . . . International Organization for Standardization
ISP . . . . . Internet Service Providers
IS . . . . . Information Systems
ITS . . . . . Incompatible Time-sharing System
J2EE . . . . . Java 2 Enterprise Edition
JPF . . . . . Java Plug-in Framework
LGPL . . . . . Lesser General Public License
LIR . . . . . Local Internet Registry
LKM . . . . . Loadable Kernel Modules
LUG . . . . . Linux User Group
MIT . . . . . Massachusetts Institute of Technology
NCP . . . . . Network Control Program
NDA . . . . . None Disclosure Agreements
NEPAD . . . . . The New Partnership for Africa's Development
Net/1 . . . . . Networking Release 1
Net/2 . . . . . Networking Release 2
NGO . . . . . Non-Governmental Organisation

NIR . . . . . National Internet Registry
NJMF . . . . . Norwegian Iron and Metal Workers' Union
NLM . . . . . Norsk Luthersk Misjonssamband
NORAD . . . . . Norwegian Agency for Development Cooperation
OI . . . . . Organisational Informatics
OLPC . . . . . One Laptop Per Child
OOP . . . . . Object Oriented Programming
OSI-stack . . . . . Open Systems Interconnection
OSI . . . . . Open Source Initiative
OS . . . . . Operating System
P2P . . . . . Peer-to-Peer
PARC . . . . . Palo Alto Research Center
PAR . . . . . Participatory Action Research
PDA . . . . . Personal Digital Assistant
PD . . . . . Participatory Design
PICTIVE Plastic Interface for Collaborative Technology Initiatives through Video Exploration
RAD . . . . . Rapid Application Development
RDP . . . . . Reconstruction and Development Program
RIR . . . . . Regional Internet Registry
SAIL . . . . . Standford University's Artificial Intelligence Laboratory
ST . . . . . Structuration Theory
TCO . . . . . Total Cost of Ownership
TED . . . . . Technology, Entertainment and Design
TPLF . . . . . Tigrayan Peoples' Liberation Front
TRIPS . . . . . Agreement on Trade-Related Aspects of Intellectual Property Rights
UEDF . . . . . United Ethiopian Democratic Forces
UiO . . . . . University of Oslo
UNMEE . . . . . United Nations Mission in Ethiopia and Eritrea
USL . . . . . Unix System Laboratories
UUCP . . . . . Unix to Unix Copy Program
VBA . . . . . Visual Basic for Applications
VCI . . . . . Virtual Community Informatics
WAN . . . . . Wide Area Network
WPE . . . . . Workers' Party of Ethiopia
WSIS . . . . . World Summit on the Information Society
WTO . . . . . World Trade Organisation
WWW . . . . . World Wide Web

A.2 List of Figures

A.3 List of Tables