IDIA International Development Informatics Association

Proceedings of the 3rd International IDIA Development Informatics Conference
28-30 October 2009
Berg-en-Dal
Kruger National Park
South Africa
ISBN 978-0-620-45037-9

Towards "best practices" in North-South open source projects - lessons learned from the ARIS project in Mozambique

M. Pscheidt
University Information Systems
Universidade Católica de Moçambique, Mozambique
mpscheidt@ucm.ac.mz
and
Institute for Computing and Information Sciences
Radboud University, Netherlands
markus.pscheidt@gmail.com

E.J. Simons
Senior staff, Concern Information Management
Radboud University, Netherlands

Th.P. van der Weide
Institute for Computing and Information Sciences
Radboud University, Netherlands
tvdw@cs.ru.nl

Abstract

In this article we present some aspects that in our view are to be taken into account in order to create successful and sustainable development and implementation of open source systems in institutions of higher learning in developing countries. The ideas and findings presented here are partly based on study of literature and partly drawn from the experience of the authors within the framework of a project to develop and implement an academic registration and information system (ARIS) for Mozambican universities.
Some of the propositions made in this article are specific for the sector or domain of higher education, while others are more generic and could possibly be used as a source of inspiration for the development and implementation of open source software systems in developing countries in general.
The objective of this paper is to show the opportunities, challenges and ways to approach information systems like ARIS with open source concepts. We show possible building blocks for open source projects, with the intention to contribute to "best practices".

Keywords: Open Source, Development cooperation, Mozambique, Higher Education, Academic Registry, Information System

1. Introduction

Free Software and Open Source software has gained momentum in developed countries. In developing countries it has been hailed as a chance to overcome specific problems in software development like (1) dependencies on foreign software vendors, (2) the inappropriateness of foreign developed software, (3) relatively little technical skills and (4) the high cost of software licenses.

However, a significant participation by developing countries in Open Source initiatives so far has been mostly related to only a few applications like Linux, Mozilla and the Apache web server. In contrast to these popular software programs, which are implemented by a high number of users, open source has less share in the area of end user applications, both in developed as in developing countries. One possible reason is that open source traditionally favors technically interested 'user-developers', i. e. those users who have technical skills to modify source code and improve the software.

Thus, despite a high theoretical potential for open source in developing countries, it seems difficult to realize this potential practically. For a collaborative endeavor such as open source software development all parts of the network need to have adequate capacity (Braa, Monteiro et al., 2007).

While many universities in developing countries suffer with difficulties related to the management of their academic data, up to now there are no open source products available for academic registry management in developing countries. Besides the open source difficulties mentioned above, we attribute this to the complexities involved with software use and development, stakeholder coordination and domain knowledge.

Using an information system to support academic registration is a common strategy for universities in developed countries. More and more, also in developing countries the use of an effective information system is seen as a 'conditio sine qua non' for success according to international standards. In order to improve management capacity at Mozambican universities, in 2005 a development cooperation project has been started. Part of this project was the development of an academic registry information system (ARIS) with the aim to provide a software system that fits the Mozambican reality, including a Portuguese user interface. Existing information systems were analyzed but no appropriate option was found that could be reused. Therefore it was decided to develop an ARIS during the project lifetime that supports the Mozambican reality in higher education. The project ends in 2009. The experiences gained during the project are used to reflect on aspects of North-South development cooperation in the area of software development. More specifically, this paper investigates the possibilities of open source software related to development cooperation and presents building blocks for the utilization of the open source methodology.

Information systems created through a development cooperation project between North and South face the challenge of being sustained after the end of the donor funds. For continued user satisfaction it is necessary to stimulate continuous development and to facilitate local adaptations e.g. to provide user interfaces in local languages and locally relevant reports. We analyze how to achieve such goals by looking into aspects such as software licensing options, stakeholder interests and coordination, and software design aspects like modularization and internationalization. The goal of the paper is to develop building blocks that can serve as rough guidelines for similar projects.

The structure is as follows. First we give some background on the ARIS project, followed by a discussion of relevant Open Source concepts. After that we elaborate a model and building blocks for open source projects. Finally, we give some conclusions.

2. Description of ARIS

A conventional Academic Registry Information System (ARIS) is used for the registration and management of student, staff and course data within a university or other institute of higher learning. In this paper we have chosen to define a generalized version of ARIS, which we base on our practical experience during a North-South cooperation project to develop an ARIS for Mozambican universities. Several local universities implement (install and use) the system, which has been developed mainly by the foreign development partner. Knowledge transfer and the creation of local ownership are goals to be achieved until the end of the development cooperation project.

The incremental development model (Davis et al., 1988) was selected as the basic software development methodology. The reason was to be able to involve local stakeholders early on during system development. After the construction of a partial system, end users and technicians have been reviewing progress and providing feedback for further system development on a regular basis.

Figure 1 shows typical stakeholders in an information systems project. While there are other groups involved, the groups of users and developers are highlighted as they are found in the various organizations. Software development takes place to a certain extent at the donor side, and local development is established during the project lifetime. This can be done by particular implementing institutions and by a local expertise center. From our experience we can say that the proximity of users and developers is a boost to innovation for the product in question. Therefore, the existence of implementing organizations that have the capacity to improve the system is an ideal combination, especially if improvements are fed back into the global source code of the software so that other organizations also receive these improvements.

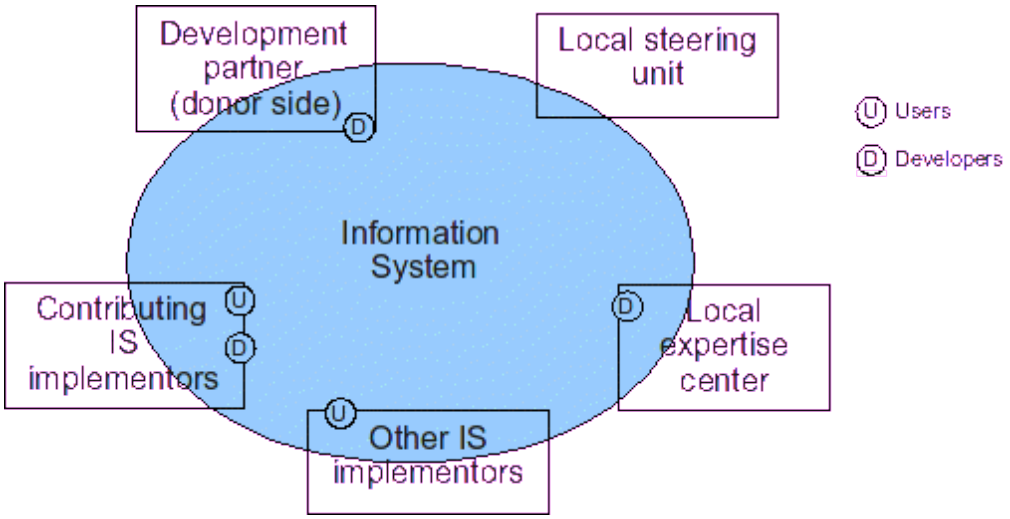

Figure 1: The stakeholders in an information system development cooperation project

Stakeholders can be further organized in the following groups, which relate to different phases in the software development life cycle.

1. Management Team: this group is responsible for overall management and financing issues. This team also handles the reporting to the funding organizations.
   In development cooperation projects this team includes both the Northern and Southern partner. Part of this team is the local steering unit. This is a subgroup to manage the local processes. There will also be a subgroup to manage the donor activities.
2. Requirements Team: this group is responsible for the formulation of the requirements (and functionality) of the system to be built. This group will mix local expertise with

international expertise. People from this team may also be characterized as domain experts.
The Requirements Team will formulate a requirements report and agree on the system model as negotiated with the Development Team on the basis of this report. Ideally, the Requirements Team has an international character.

3. Operations Team: this group is responsible for the operations of the (new) system. This group is aware of the local situation and can transform the local situation requirements into constraints and/or parameter settings of the Information System (IS). This group also provides the local infrastructure required for the IS (such as for example setting up a help-desk).
This group preferably should consist of local participants. This will require that universities will adopt in their programs the opportunity to build these skills. Until that time, international partners can assist the Operations Team.
4. Development Team: this group is responsible for modeling the requirements from the Requirements Team, and to validate this model with this team. Then this team will set up a system design in accordance with the Operations Team. This team will consist of the donor development partner, the local development team and maybe some subcontractor. The Development Team also is responsible for the implementation of the system and to carry over to the Operations Team.
The Development Team has a local component that handles the reception of the new technology as set up by the donor partner. This component may also have been organized as a local community in the open source sense.

Note that capacity building is an important issue during the project, to ensure skilled people are available for the Development Team. Only that way the local contribution can cope with an exposure to the open source philosophy.

Until that time, the Development Team may insource from the international partner. We see a relation with outsourcing activities. Outsourcing experiences may be helpful to improve the success of insourcing.

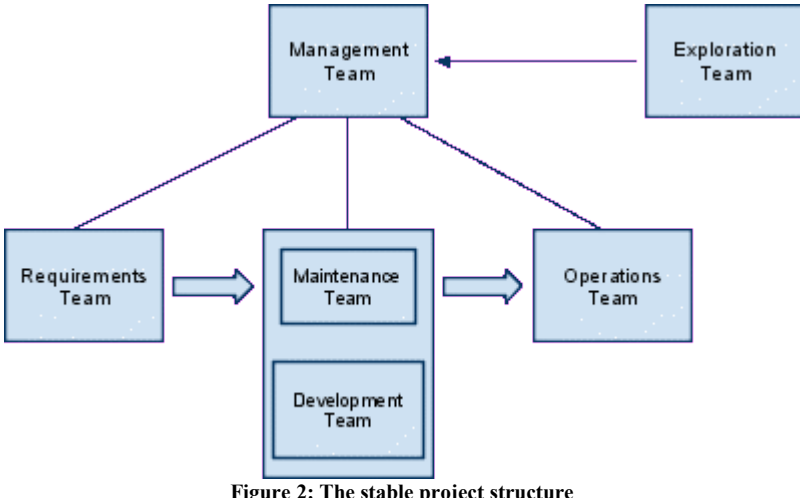

Figure 2: The stable project structure

In a later phase, when there is a running product, an extra team may be added to the product community:

5. Exploration Team: this team combines and shares best practices, and finds to opportunities to explore the system optimally. Typically this team will consist of user groups.

Furthermore, there will also be a special team for maintenance. In many projects the Maintenance Team by far is the largest team. The idea behind Open Source is to share maintenance and development. Moreover, the open discussion is expected to lead to a more stable architecture of the system.

We want to emphasize the important role of the Management Team. It is responsible for coordination between the other groups and to react to often dynamic realities. Furthermore, the exploration team has an important role to play to highlight upcoming requirements. These requirements need to be prioritized and given to the other teams for system improvement. See Figure 2 for an overview of the project teams. Note that the teams vaguely reflect the stages in the software development life cycle.

3. Theoretical considerations

The main objective is to sustain the information system after the end of the developing cooperation project funds. Aspects of this overall goal include:

- to ensure system maintenance at a low level,
- to ensure capacity building to do maintenance and further development,
- to stimulate continuous development and adaptation, since a stop in continuous development leads to a degradation in user satisfaction (Lehman & Ramil, 2001),
- to enable community building by currently involved and potential future stakeholders,
- to support the adaptability of the system to the needs of implementing universities,
- to provide a support entity to which users and organizations can turn to in case of difficulties,
- to foster local ownership.

The analysis will take into account issues including the following:

- software licensing in general and open source licensing in particular,
- stakeholders and their interests,
- support of users and other parties,
- design aspects of the software concerning modularization and internationalization,
- motivation factors for participation in OSS on individual and institutional level,
- control regarding source code commits to maintain source code quality.

There are several levels that need to be addressed: the viewpoint of a particular university, the Mozambican community that is formed through the common use of and interest in the same software, and thirdly the international level that includes the development cooperation partners and potential future participating universities that may get interested in the system if it is made publicly available.

Based on these initial observations, the remainder of this section discusses theoretical concepts that will be used to elaborate open source building blocks in the following section.

3.1 Software licensing

In many countries source code is protected by copyright and intellectual property regulations. As soon as source code is created, it is protected by copyright. Therefore, source code cannot be used legally by other people than its creator if she doesn't grant such rights to others. Software licenses define which rights are given to other users. The spectrum of licensing options ranges from public domain to open source licenses and to commercial licenses.

When an author puts his source code in the public domain, others are allowed to use it however they like. With commercial licenses on the contrary, the allowed software use is accurately specified and limited, e.g. to the right to install and run a single copy. The user

might also be limited by a time-based or seat-based scheme. Open source licensing can be placed in the middle of the range and consists itself of a spectrum of licenses on its own and ranges from academic and permissive to restrictive licenses. See Table 1 for an overview of types of licenses.

Table 1: Recommended open source software licenses (Lindberg, 2008)

Type of license	Recommended licenses
Academic licenses	2-clause BSD license
Permissive licenses	Apache License version 2.0
Partially closable licenses	Mozilla Public License Lesser GPL (LGPL) version 2 or 3
Reciprocal licenses ("viral" licenses")	GNU GPL version 2 or 3 Open Software License (OSL)

Academic and permissive licenses give a lot of freedom to the use of the source code, but protect the authors from being legally challenged by users, e. g. to protect trademarks like names and logos. Academic and permissive licenses allow products derived from open source code to be redistributed in a closed source fashion. Examples for academic licenses are MIT and BSD licenses. An example of a permissive license is the Apache V2.0 license.

Partially closable licenses allow certain parts of the code to be distributed freely, e.g. commercially, while other parts of the code are required to stay open source. Because of this separation, partially closable licenses usually show up in two areas: libraries and extensible applications. For example, libraries licensed under the LGPL can be redistributed in proprietary applications in unmodified form, but any changes to the source code of LGPLed code have to be distributed. Applications that include LGPLed libraries are not required to apply open source licensing to the whole application. On the other hand, extensible applications have an open source core and allow proprietary extensions.

Reciprocal licenses require that binary distributions include the full source code of the application. They are sometimes called "viral" licenses because of the requirement that applications need to apply open source licensing on the whole product if any source code is included that is licensed under a reciprocal license.

Because of different restrictions the compatibility of licenses is limited. If a given software system is licensed under the LGPL it may be redistributed in a software under the GPL, but not the other way around. Another aspect of license compatibility is related to the big number of different licenses. These may not be compatible with each other, and it may discourage possible contributors to use less popular licenses. Lindberg (2008) recommends to limit the choice of open source licenses to a small set that should be sufficient for the majority of circumstances (see Table 1).

Lerner & Tirole (2005) analyzed projects hosted on Sourceforge and made the following observations about license choice:

- Restrictive licenses are more common in projects geared towards end users, like desktop applications, in languages other than English, and designed for a non-commercial user environment or operating system.
- Community contributions are greater when restrictive licenses are employed.
- Software that is made available under nonrestrictive licenses is particularly prone to "hijacking" by commercial software vendors, meaning that somebody may add some proprietary code to the software and take the whole private.

- A more restrictive license reduces the opportunity for making money on complementary products.

"Free software" and "Open Source" have different origins, and not all software licenses that are recognized as open source licenses are recognized as free software licenses and vice versa. Despite ideological differences many developers feel little impact on the actual software development in their communities (Eilhard, 2009). In this paper we use the term "open source software" to refer broadly to free, libre and open source software. It shall be emphasized that free software is not be confused with freeware. As Richard Stallman puts it, free software is "free as in free speech, not as in free beer" (Gay, 2002).

3.2 Opportunities, challenges and culture of open source

Commonly stated advantages of open source in relation to closed source include robustness of the code, flexibility to the user (e. g. avoid vendor lock-in) and support from a community (Krishnamurthy, 2003).

One particular challenge associated with open source software is 'forking', the development of different, eventually competing, versions of the same software. This happens because of different opinions of developers. While commercial software production follows hierarchical structures, open source software production does not have an obvious method to resolve disputes. This indicates the importance of sound governance of open source projects (Kogut & Metiu, 2001).

Many open source advocates argue that open source software tends to be geared towards the more sophisticated users, such as user-developers. In comparison to commercial software there is less emphasis on documentation, support, user interfaces and backward compatibility. The greatest diffusion of open source projects appears to be in settings with sophisticated end users such as Apache servers (Lerner & Tirole, 2002).

Contributors to open source software have a culture of "maximizing reputation incentives", of "ensuring that peer credit goes where it is due and does not go where it is not due". Amongst open source contributors exist three taboos to avoid damage to their reputation: (1) forking of projects, (2) distributing rogue patches and (3) "surreptitiously filing someone's name off a project". The latter is considered an ultimate crime (Raymond, 2000).

Braa et al. (2007) outline opportunities and challenges in open source software development within a South-South-North network: "Sharing of resources is one of the great promises of Free and Open Source Software (FOSS) approaches to software development – but it also puts demands on the local capacity in all parts of the network, ranging from software development to adapting the use context – capacity to both meet local needs and contribute to global development at the same time."

3.3 Community initiated vs. spinout projects

Open Source projects are created in different ways: either as "community initiated" projects or by releasing code that has been internally developed by a sponsor. The latter is called a "spinout" project. Spinout projects start off with a software system that typically is already usable, while community initiated projects start from scratch. Community initiated and spinout projects have different reasons for initiation, key issues, motivation for contribution and control mechanisms. West & O'Mahony (2005) have summarized these issues, as shown in Table 2.

Table 2: Comparison of community initiated and spinout projects (West & O'Mahony, 2005)

	Reason for initiation	Key issues	Contributor motivation	Control
Community initiated	- Solve a problem - Create a "free software" alternative to proprietary solution	- Gathering Resources - Building healthy community, attracting talented developers - Distributing software - Gaining "mindshare" with minimal marketing	- To make software happen - To gain fulfillment - To build and learn new skills - To solve personal and professional problems	- Democratic, transparent, usually meritocratic - Some leadership and stratification
Spinout	- Achieve greater adoption - Get development help on areas that are of low priority for the firm (e. g. special dialects)	- Gaining legitimacy Building healthy community, attracting talented contributors - Resolving ambiguity about control and ownership	- To complete areas that are of high priority for contributors - To gain visibility by prospective employers - To influence sponsor's alignment with complementary projects	- Varies but usually sponsor retains direct or indirect control

3.4 Economic sustainability

The philosophies between free software and the free market are quite different. Open source software is related to cooperation, inclusion, sharing and openness, while the market is related to competition and self-interest. Chege (2008) defines "software dialectic" as the challenge of finding a way to ensure economic sustainability without sacrificing the ideals of free software.

The recent popularity of more liberal licenses in relation to the GNU GPL license shows a pragmatism to combine the advantages of open source with for-profit activities. Historically the greatest diffusion of open source has been in settings where end users are sophisticated, such as system administrators. These advanced 'user-developers' tolerated lack of detailed documentation or easy-to-understand user interfaces. With increased business focus open source software enters segments that were traditionally poorly served by open source software (Lerner & Tirole, 2002).

Richard Stallman, the president of the Free Software Foundation (FSF): "We encourage people who redistribute free software to charge as much as they wish or can. [...] Distributing free software is an opportunity to raise funds for development. Don't waste it!" (Gay, 2002) While there is no objection to earn money with software, the GNU GPL puts limitations how to use the software licensed with it. Therefore business with GPLed software is hardly done with selling the source code unless the software developers are contracted by someone for their job. Profit is rather made by third parties that sell services around the software, like training, support, distribution, hosting and consulting. GPLed software is typically geared towards sophisticated users. With more permissive licenses (see table 1), products derived from the open source software can be redistributed in closed source fashion. This creates possibilities for income (Krishnamurthy, 2003).

3.5 Governance

Leadership in Open Source projects is an important determent of project success. A common feature of many open source project leaders is that leaders are programmers who made important contributions early in the project's development, and moved on to broader project management tasks. The leader in an open source setting often has no formal authority over programmers, but her recommendations tend to be followed by the majority of programmers in the project. Leadership has an important role to play in accepting or rejecting modifications to the code in order to keep a required level of quality, especially because of the absence of liability as would be the case if sold by a commercial software firm. The key to successful leadership is the trust by the programmers (Lerner & Tirole, 2002).

The benefits of the open source production method are limited by the quality of the coordination process, by the level of redundancy of development and by versioning problems (Kala, 2008). Bad governance can result in delays, redundant production or the forking of the project. Effective governance contains free-riding, coordinates the software development and ensures the quality of the outcome. There are three typical kinds of leadership in open source projects: charismatic leader, voting committee and rotating leadership (Eilhard, 2009).

Shah (2006) compares an open source project with a so called "gated source" project, which is an example of a tightly controlled company sponsored project that seeks volunteers but limits the granted rights given to the volunteers; the source code stays in the possession of the company to make profits by selling commercial licenses. Developers generally are suspicious to corporate run projects, because contributions can be hijacked for company profits without getting recognition for their contribution. To overcome reservations, companies need to interact carefully with the community: (1) involve private contributors in the decision-making process, (2) use a restrictive open source license that effectively prevents corporate code hijacking, or (3) hire renowned open source developers to show the commitment to the open source idea.

3.6 Favorable project characteristics

Favorable characteristics for open source production include project modularity, existence of fun challenges, credible leadership that provides vision and keeps the project together, i.e. prevent forks. At the start of the open source project enough code shall be present so that a community can react as well as get convinced that the project has merit. The aspect of modularity is considered essential for the viability of open source software. Baldwin & Clark (2003) argue that the architecture, particularly modularity, is a critical factor. And Kogut & Metiu (2001) even declare that a product that is not modular is not considered appropriate for open source development.

3.7 Open source in the developing country context

Walsham & Sahay (2006) describe open source software as a relevant technology in the context of developing countries and emphasize the technological details of open source licensing agreements.

There is considerable interest about using open source software in developing countries, but there are different opinions if and how it can be useful. Heeks (2005) states a lack of strong evidence of FOSS benefits and questions it to be a blind alley for developing countries. A more optimistic view is expressed by Lerner & Tirole (2002): "Users in less developed countries undoubtedly benefit from access to free software" (p. 198). Walsham & Sahay (2006) consider open source software a relevant technology in the context of developing countries and emphasize the technological details of open source licensing agreements. Given

such a variety of opinions in the following we argue that there are good reasons to consider open source software in developing countries, that the benefits outweigh the con's.

On the governmental level, developing countries have begun to embrace open source motivated by

- a desire for independence,
- a drive for security and autonomy, and
- new intellectual property rights enforcement and productivity (Weber, 2004).

Countries around the world try to minimize their reliance on single suppliers who may not be focused on the country's interests, and to avoid opportunism's by suppliers because of vendor lock-in. Open source represents a possible route for more local participation in software development, thereby contributing to a local software industry, keeping expenditures within the region and effectively improving local independence.

In an attempt to highlight problems with proprietary software, Peruvian Congressman Edgar Villanueva in an open letter to Microsoft Peru objected the monopoly of data encoding and processing by a single provider, and argued that the usability and maintenance of software should not depend on the goodwill of suppliers or monopoly conditions imposed by them. Other developing countries have also expressed grievances with proprietary software, pointing to the little influence they have as small customers on how software develops. Open source is expected to provide more flexibility and to allow more autonomous input, fostering local ownership. With open source an indigenous industry can potentially participate in both identifying and meeting software development needs.

With increasing attempts to fight software piracy in developing countries open source represents a strategy to comply with intellectual property regimes. But there is a deeper issue involved. The degree to which a software tool can be utilized and expanded is limited only by the knowledge and innovative energy of the users, not by exclusionary property rights, prices and the power of corporations. Free access to source code not simply costs less but enables learning by doing, which creates demand and can lead to the development of applications that fit specifically indigenous needs.

Ghosh (2004) gives three main reasons for using open source software in developing countries:

- cost (the total cost of ownership),
- performance, flexibility, localization,
- skills development.

Open source software helps localization; proprietary vendors do not care particularly about local issues and consequently local issues are ignored, whereas open source software makes local adaptations possible. As Ghosh puts it by example: "Many FLOSS developers may have absolutely no interest in software usability for Xhosa speakers. But FLOSS developers allow and encourage those with locally relevant motives to adapt their software."

Open source software develops local skills; not the skills to use the free software applications, but skills like programming and teamwork. These skills are learned by participating in the open source community. Instead of only passively using the system it allows active creative productive work. This is important since skill development requires access to the ability to create. Ghosh deducts that this low entry barrier to being creative enables a technology transfer mechanism from those in the open source community who have knowledge to those who do not. If open source is applied to North-South cooperation’s technology transfer can be integrated as an integral part of such projects.

3.8 Technology transfer

Technology transfer between developed and developing countries mainly occurs through direct import of technology, e.g. by purchasing equipment. While this can give quick results, little knowledge is transferred that would lead to independent production and customization of technology. More generally, typical problems of technology transfer to developing countries are:

- Asymmetric information: the knowledge holder does not reveal information without incentives, and knowledge receivers cannot identify the value of information before buying it.
- Market power: technology owners are interested in profits and in covering costs of innovation processes.
- Limited movement of people: in developing countries incentives are typically too weak to support free movement of people and their knowledge.
- Intellectual property rights: costs of licenses and other fees (Alkhatib et al., 2008).

This characterization of problems might suggest that technology transfer has a unidimensional character, that developed countries are the sources and developing countries the recipients of information. But technology transfer is not a one-way information flow but a two-way communication process. Transfer is the communication of information. One should think of participants in the technology transfer process rather than 'sources' and 'receivers'. Each party involved in the technology transfer process may have a different perception of the technology. These differences may be worked out through two-way communication (Rogers, 2002).

Open source is based on direct communication. It allows direct access to the knowledge source. Local innovation is not limited by intellectual property restrictions. In turn, the viability of local innovation reduces the brain drain.

Open Source can be used as a tool to make the step from exclusively foreign system development to the involvement of local developers in developing countries. It allows local learning and better coordination between the two sides in development cooperation projects, because code is shared and local developers can be given responsibility according to their current level of knowledge and increase their knowledge level.

4. Building blocks for best practice open source development

Based on the experiences of the ARIS project, its generalization and the theoretical considerations, in this section we will formulate a model with building blocks for open source based development cooperation projects.

An objective of the model is to handle the actual reality in terms of local capacity, i.e. to develop local capacity to required levels over time and to substitute the missing capacity while it is not yet available locally. The actual level of local capacity may vary in different local implementing organizations, and over time, e.g. due to staff fluctuations. Therefore it is desired that local implementations are not jeopardized because of the reality of local capacity. The model presented in the following tries to achieve this by making the software development process a joint effort between participants in developed and developing countries and facilitating local learning during the process.

4.1 Model overview

The conceptual model (see Figure 3) is based on the distinction between the global software core and localized versions of the software. The core is the common building ground for all implementing organizations. It resembles the functionality that is considered useful for most instances of the software. Localized versions are derived from the core through local adaptation and enhancement.

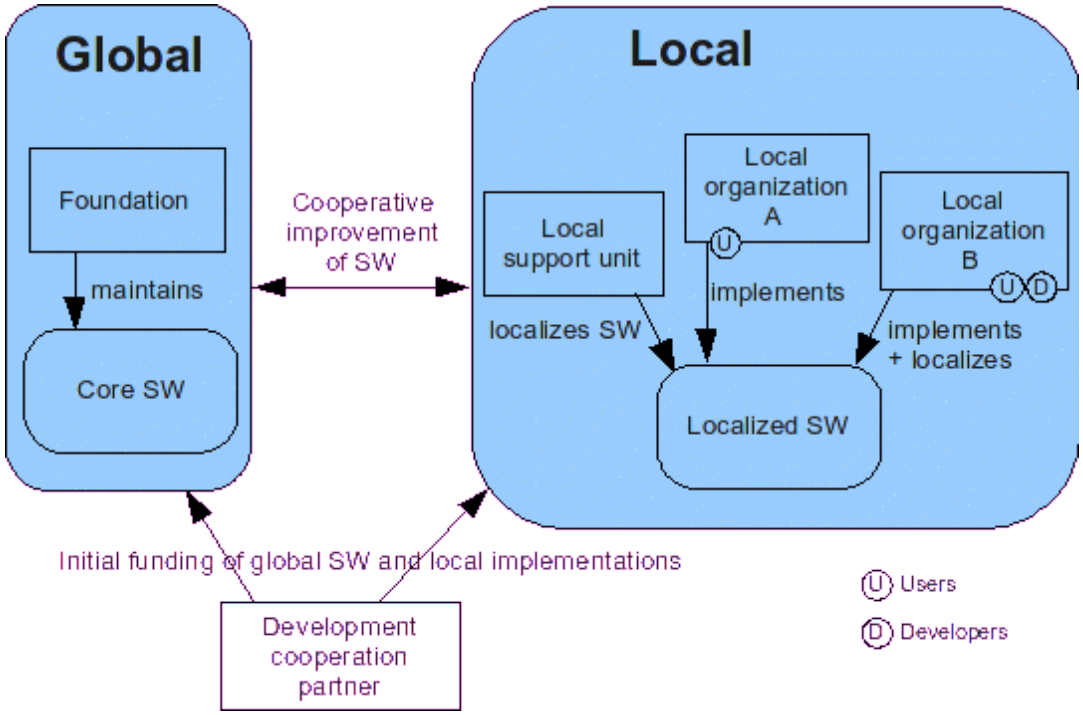

Figure 3: Conceptual model for open source development

The model's central focus is on local users and on local capacity building. To realize the intended benefits in these areas it is required to build proper communication and coordination skills. Here the development cooperation partner comes in to cooperatively set up the structures, to encourage proper communication and to build the basic skills. An important element in this process is the implementation of the system in an organization in the sense as Walsham (2009) describes it: implementation in a human and social sense, so that the system is used frequently by organization members or that it is considered valuable for work activities or coordination. The implementation effort helps to discover practical difficulties and to find ways to fit the technology with local needs. The particular relevance of practical implementation for local learning is stated by Braa, Monteiro et al. (2007) in which they describe it as "arguably the most effective learning mechanism" in the context of district health information systems in several developing countries.

There is supposed to be one single core system and many local systems. The core system is managed by a dedicated legal entity. Local organizations will first evaluate the system and eventually decide to introduce it into their organization. They need to go through an implementation process that consists of such activities as installation, configuring organization specific properties of the system, importing existing data and aligning internal processes. There may be certain requirements that are not covered by the given core system, and local enhancements will be desirable or necessary. In this case the implementing organization may use internal capacity or contract a third party to develop the required enhancements.

When a local side enhances the functionality of the system, e.g. by developing an additional module that can be plugged into the system, then potentially this enhancement can be integrated into the core of the system. Vice versa, when the core software is improved, it can be used to upgrade local implementations. Therefore, the system can cooperatively be improved by the local implementers and the maintainers of the core system. To find the boundary between core functionality and localized functionality is a tough decision, though, and requires properly skilled staff.

A good balance between core and localization furthermore minimizes the tendency to fork: local implementers may be inclined to change source code belonging to the core system to suit own needs, for example because of time pressure during the local implementation phase. But if these alterations are not accepted to be incorporated in the global core then a fork is created that limits the possibility to stay in sync with the global core and take advantage of its improvement at the local site. Therefore, communication and proper care are important from both sides.

The existence of an organization managing the global core of the software provides the basis for development cooperation and technology transfer. It is the means to avoid that local sides are left alone after the end of development cooperation projects. One could argue that the creation of a central organization in fact creates dependencies from developing countries to the developed world. But evidence shows that the gaps between required and existing capacity are often too big in developing countries to be bridged in the short term (Heeks, 2002). Therefore two different phenomenons need to be distinguished: On the one hand dependency of the central maintenance unit is created. On the other hand a structure is provided to support developing country partners and to facilitate local capacity building. Therefore the model is designed to foster stable operation of the information system in implementing organizations and to allow equal partnership with both sides contributing and advancing the system.

Open source offers the possibility for user centered innovation. Local software development is closer to the end users than software that is produced in developed countries and afterwards exported. End users of application software can rarely be expected to be 'developer-users' themselves, particularly in developing countries. But we consider a certain closeness of developers to end users essential to streamline needs with software development and thereby facilitate the usability of the software.

4.1.1 Financial sustainability

The issue of generating funds to sustain continued evolution of the software system is related to the roles the different stakeholders play. The existence of the central unit is motivated by the fact that certain skills are locally unavailable. Local support units are included in the model to provide services to local organizations that (wish to) use the system. The needs that arise by users and managers in local organizations shall be served by a multiple levels of support, which comprise (1) organization internal support, (2) a local support unit and (3) the central unit. While the goal is to solve as much as possible locally, it is likely that the central unit will be needed on a regular basis. This collaboration forms the basis for financial compensation.

4.2 Building blocks

Table 3 shows an overview of building blocks for open source based development cooperations. These tools and processes are complementary to general open source concepts such as source code repositories, issue trackers and contributor recognition. The collection tries to react to the traditional weak points of open source software such as little documentation and professional support.

Table 3: Building blocks for open source information system development

Definition	
Open source project creation and control	Spinout Directed open source
License	Depends on application type
Agreements	Contractual agreements with local organizations to secure involvement of developers and project leaders from local organizations with domain knowledge
Construction	
Development model	North-South-South: core lead group of experts building the initial software kernel in the North, increasing involvement and responsibility in the South through module and report development, implementation and support
Architecture	Modular software structure Build upon open source frameworks and libraries
Implementation	
Development focus at implementation	Reserve resources for development during implementation at organizations to deal with upcoming issues
"Concentrated implementation package"	Apply a package of activities for the practical implementation of the system
Operation	
Local support structure	Multi-level support with emphasis on solving problems close to its source
Trust building	Foster an atmosphere of trust Active communication culture Take concerns seriously

4.2.1 Project creation

The initiation of a development cooperation project follows the formulation of a need of stakeholders in the developing country. Under the spin-out model first a working version of the software is developed before the source code is published under an open source license. During the initial development the focus is to start a community of the developers in the development cooperation project, both from the North and from the South. When the software advances, spinning out the source code can lead firstly to greater adoption of the software by organizations that have initially not been in the project, and secondly to a more stable community. Furthermore, it can help to resolve ambiguity about control and ownership by applying open source principles. The control typically retains to a varying amount with a central entity. Courant & Griffith (2006) call such centrally controlled projects 'directed open source' projects. See table 2 for a summary of characteristics of spin out projects.

4.2.2 License

While the particular license choice depends on the context, some indications can be given. In many cases the license attributed will need to take into account both community interests and commercial interests. A careful trade-off between the two is necessary in order to indicate proper license options. Community interests include the attractiveness of the software and the adaptability to local needs. Commercial interests have to be satisfied in order to ensure the funds for continuous development.

If funds need to be earned to fund further development of the software then a 'partially closable license' may be a good choice to combine advantages of open source with commercial interests. This possibility is particularly relevant with a modular system architecture. For recommended licenses see Table 1.

4.2.3 Agreements

Practical creative work is an effective method for learning. Therefore local software developers and domain experts shall get involved as early as possible. From our project experience we noted that local involvement can be inhibited because of a common attitude towards development cooperation projects: Because of a tendency to consider the project partner from the North as active producers and givers, many see the partner from the South as passive recipients. This view often prevents active engagement from all participants, and promises of involvement by local organizations are often not kept. This vicious cycle needs to be avoided to give technology transfer a chance.

Before the initiation of software development activities, agreements shall be formalized that will give an indication that local learning is realistic during development cooperation project lifetime. More concretely, agreements between institutions and their developers and domain experts are advisable to guarantee their availability for project activities. This also prevents investment in training of developers who will ultimately not be available to use their knowledge.

4.2.4 Development model

The development model tries to combine two goals: Develop a high quality software and local capacity development.

- Project phase: core lead group of experts (2-5) from the North building the kernel of the application with gradual introduction of developers from the developing country, more specifically through development of separate modules. After 'proven competence' in the additional module development, they can participate in the core development.
- Operation phase: NSS-model (North-South-South), with North still leading and guarding the core development, but southern partners leading additional module development and taking care of support and major part in the implementation activities also in other southern countries starting to implement the system.
- In the long term, lead development is not bound to the North. Coordination shall develop through open source dynamics, built on experience gained by project contributors.

For the realization of a software project, typically not only source code in the sense of programming output is required. Internationalization and reporting can be important aspects. Translation and report creation are possible entry points for initially inexperienced software developers and non-programmers. The steps of the ladder of increasing knowledge and local ownership can look like:

1. local developer is able to install the system, to give basic technical support to users
2. local developer is able to build the system from a central repository
3. local developer is able to do simple technical tasks, for example
    - adaptations for local organization
    - graphical report design
4. local developer takes part in a team to develop certain functionality or module
5. local developer is the 'owner' (i.e. responsible) of a module
6. local developer has the competency and is trusted to improve the kernel of the IS

The knowledge increase of local developers is possible by experience and feedback by fellow developers.

4.2.5 Architecture

The architecture is based on a set of independent modules dealing with separate sections of functionality, including a core module dealing with the generic functions for the application domain. OSGi (www.osgi.org) is an example of a framework that provides advanced modularity. The extent of the core module typically will be limited to functionality that is uniform for all implementations. An example of a module with separate functionality is the reporting part, dealing with the output of the system.

Another, somewhat obvious, architectural element is to base the system on open source components. The license of the used components will have an influence on the possible license to use for the developed system.

4.2.6 Development focus at implementation

The implementation of the information system at an organization is meant here not only to install the system, but also to streamline the organization's work processes and to reach the point that users consider the system an added value. Depending on the users' computer skills, domain skills and other factors the implementation can take considerable time. One of the factors is the maturity of the system itself. During the implementation phase emerge program bugs, unhandy behavior or other issues that will require improvements to the system to be acceptable in the work environment. Therefore, some extra development efforts shall be planned for the implementation phase. For improvements resulting from implementation experiences, like in all further development, the careful consideration has to be carried out between improvements in the global or adaptations in the local version of the source code.

4.2.7 Concentrated implementation package (CIP)

To point to the significance of the implementation phase for local learning and acceptance of the system in the target organizations, we want to recommend a package of activities and products that together form a 'concentrated implementation package':

- Required hardware and software configuration to run the system
- The core system itself and the available additional modules
- Full user and technical documentation in electronic form, but also copies of each in printed form, given their shortage of printing material
- Detailed scenario for the technical installation (description of steps, people to involve, time planning)
- Intensive user training
- Idem for the technical support people of the system (server administrator, system management)
- Detailed description of the 'best practice' workflow, taking into account the specific context (previous way of working, organization) of the institution
- Training of the management concerning the effects and consequences of the implementation of the system for their organization
- Import of existing data

The CIP should be worked out by the central leading authority, in consultation with local representatives of the institution and be executed by the NSS construction.

4.2.8 Local support structure

The local support unit has the objectives to offer training on the use and technical maintenance of the system, to coordinate evolving requirements with software developers, to be a point of reference for new organizations that are interested in the implementation of the system. The local support unit is part of a multi-level support structure. Most emerging problems will be solved inside the organization. Some will need to be answered by the local support unit. A few rare cases will be answered centrally.

4.2.9 Trust building

The interaction between different stakeholders like users and support center requires a certain level of trust. While it takes a long time to accumulate trust, it can be quickly lost by e. g. disappointing stakeholders or simply ignoring their requests. Therefore a communication culture should be fostered that, at a minimum level, includes to react to request and give feedback.

Advancing from this basis, actively approaching users and managers of implementing organizations will give a valuable picture of current satisfaction and challenges.

5. Conclusions

This is a work in progress. There are few studies that add to the body of experience on the joint development of information systems in North-South collaborations. In this paper we have contributed to possible best practices for such collaborations within the open source frame. We built upon literature and our experience gained in the development cooperation project about the provision of an academic registry software for Mozambican higher education institutions.

One line for further research is to apply the criteria of Appropriate Technology (van Reijswoud, 2009) to the building blocks developed in this paper, and to formulate tools and processes that fit into the Appropriate ICT framework.

References

Alkhatib, J., Anis, M., & Noori, H. (2008). Open Source: The next big thing in technology transfer to developing nations. *International Association for Management of Technology IAMOT 2008 Proceedings.*

Baldwin, C. Y., & Clark, K. B. (2003). *Does Code Architecture Mitigate Free Riding in the Open Source Development Model?* (Working paper). Harvard Business School.

Braa, J., Monteiro, E., Sahay, S., et al. (2007). Scaling up local learning - experiences from South-South-North networks of shared software development. *Proceedings of the 9th International Conference on Social Implications of Computers in Developing Countries.* São Paolo, Brazil.

Chege, M. (2008). *Ubuntuism, Commodification, and the Software Dialectic*. Retrieved May 24, 2009, from SSRN website: http://ssrn.com/abstract=1309505

Courant, P. N., & Griffith, R. J. (2006). *Software and Collaboration in Higher Education: A Study of Open Source Software*. Retrieved May 30, 2009 from http://e-learningcentre.co.uk/eclipse/Resources/OOSS_Report_FINAL.pdf

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering*, *14*(10), 1453-1461.

Eilhard, J. (2009). Open Source Incorporated. Retrieved May 24, 2009, from SSRN website: http://ssrn.com/abstract=1360604

Gay, J. (Ed.) (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Boston: GNU Press. Retrieved May 24, 2009 from http://www.gnu.org/philosophy/fsfs/rms-essays.pdf

Ghosh, R. (2004). *The Economics of Free Software - and why it matters for developing countries*. Retrieved May 24, 2009 from http://www.infonomics.nl/FLOSS/papers.htm

Heeks, R. (2005). *Free and Open Source Software: A Blind Alley for Developing Countries?* (eDevelopment Briefing No. 1). Development Informatics Group, University of Manchester.

Kala, R. (2008). *The Open and Closed Styles of Talent Acquisition and Management: Which takes an edge where*. Retrieved May 24, 2009, from MIT free / open source online papers: http://opensource.mit.edu/online_papers.php

Kogut, B., & Metiu, A. 2001. Open-source software development and distributed innovation. *Oxford Review of Economic Policy, 17*(2), 248-264.

Krishnamurthy, S. (2003). *An Analysis of Open Source Business Models*. Retrieved May 24, 2009, from SSRN website: http://ssrn.com/abstract=650001

Lehman, M.M., & Ramil, J.F. (2001). Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engingeering, 11*(1), 15-44.

Lerner, J., & Tirole, J. (2002). Some simple economics of Open Source. *The Journal of Industrial Economics, 50*(2), 197-234.

Lerner, J., & Tirole, J. (2005). The Scope of Open Source Licensing. *Journal of Law Economics & Organization, 21*(1), 20-56.

Lindberg, V. (2008). *Intellectual Property and Open Source*. O'Reilly Media.

Raymond, E.S. (2000). Homesteading the Noosphere. Retrieved May 24, 2009 from http://catb.org/~esr/writings/homesteading/homesteading/

Rogers, E. M. (2002). The nature of technology transfer. *Science Communication, 23*(3), 323-341.

Shah, S. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science, 52*(7), 1000–1014.

Stallman, R. (2007). Why "Open Source" misses the point of Free Software. Retrieved May 21, 2009 from http://www.gnu.org/philosophy/open-source-misses-the-point.html

Van Reijswoud, V. (2009). Appropriate ICT as a Tool to Increase Effectiveness in ICT4D: Theoretical considerations and illustrating cases. *Electronic Journal of Information Systems in Developing Countries, 38*(9), 1-18.

Walsham, G. (2009). *Interpreting Information Systems in Organizations*. Retrieved May 25, 2009, from the Global Text Project website: http://globaltext.org/books

Walsham, G., & Sahay, S. (2006). Research on Information Systems in Developing Countries: Current Landscape and Future Prospects. *Information Technology for Development, 12*(1), 7–24.

Weber. S. (2004). *Open Source Software in Developing Economies*. Retrieved May 24, 2009 from http://www.ssrc.org/programs/itic/publications/ITST_materials/webernote2.pdf

West, J., & O'Mahoney, S. (2005). Contrasting Community Building in Sponsored and Community Founded Open Source Projects. *Proceedings of the 38th Annual Hawai'i International Conference on System Sciences HICSS '05* (pp. 196c).