Organising support on open source software in government organisations

Daniël Vijge

Organising support on open source software in government organisations

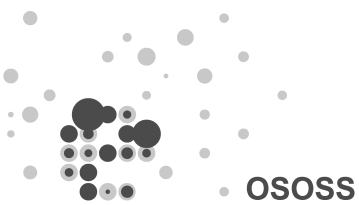
Daniël Vijge

November 2007

Master's thesis
D.S. Vijge, student number 0502654
Technology and Policy programme
Department of Technology Management
Eindhoven University of Technology

In cooperation with OSOSS, The Hague
(Open Source als Onderdeel van uw Software Strategie)

Supervised by:
dr. Önder Nomaler (Eindhoven University of Technology)
mr. Wim Wenselaar (Eindhoven University of Technology)
drs. Maarten Wijnen-Meijer (OSOSS)



TU/e Technische Universiteit Eindhoven University of Technology

Preface

For you reader the first thing to read, for me this preface is the last thing I write. It concludes nearly a year of planning, reading, research, interviewing, and of course writing. It was certainly an interesting time.

I did this report for OSOSS (Open Source als Onderdeel van uw Software Strategie), the central government agency for the Dutch government to promote open source software within the government. During my studies I became interested in open source. Not so much in the software itself, as in the mechanisms behind it. How does open source work? Why would anyone be willing to invest time in it? Should the government use open source software? I had experience with many open source projects, participated in some communities as a user, and wrote some very simple patches for some projects.

During the past year I read a lot about open source, and heard a lot about government policy while working in The Hague at OSOSS. This was certainly very interesting. Open source software is somewhat of a "hot issue". Not only for the government, but also for scientists. Many of the research on open source was conducted in the last few years, and new, interesting research is still being conducted. Many government organisations are only now getting actively interested in open source. Seeing this develop was one of the more interesting things about this whole experience.

Now that this report is finally finished I feel relieved. While it was a very interesting learning experience, it also thought me what I do not like: writing big reports. In the end I am amazed I produced a one hundred page report. Nevertheless I finished it, and can only hope you find it an interesting read.

To conclude I want to thank a number of people without whom this report would not have been possible. First of all, all the people who participated in my interviews. Then my supervisors Maarten, Önder, and Wim for giving me very appreciated and constructive feedback. And my colleagues at OSOSS, Jan Willem, Bouke, Fabrice, and Tom. I also want to thank Corence, Stein, Matthijs, and Inge for proof reading parts of my report. Finally, all my fellow graduates in the K-wing and IPO who (most of the time) were willing to have a tea break. Working hard is OK, but you should always take some time to relax and have some fun!

Daniël Vijge
October 2007

Summary

This report explores how government organisations organise the support on the open source software they use. This is done using a field study of nine open source projects, mostly within the Dutch government. By studying these projects, it is discovered how Dutch government organisations deal with open source software in general, the support on said software, and communities around these projects, to give policy recommendations to improve support.

Support for any software is important in any organisation. Software must work, but should also be easy to manage. Details of what actions must be performed for complete management and support of software applications are detailed in the ASL framework (Application Services Library). This framework consists of three layers on which applications should be managed: the strategical level, the tactical level, and the operational level.

Open source software is gaining the interest of many government organisations. But support on this type of software is often badly understood by many government organisations. Open source software can be freely used, adapted, redistributed, shared between many organisations, and can be supported by no commercial vendor al all, or multiple vendors at the same time. This creates uncertainty.

The ASL framework gives hints on how to support open source software, or actually any software. Most actions detailed in the framework are actions that must be preformed by the organisation. However, one of the actions in the framework is *development and maintenance*. Especially on this point organisations can cooperate. A government organisation can do part of the development, while other organisations, commercial companies or individuals do other parts. Collectively they form the community around an open source project. The ASL framework links the actions needed for successful support in organisations with the need for a community for an open source project.

The nine projects that are studied in this report are very diverse. Some projects are just beginning, while others are already a success. However, all projects produce their own code, either writing programs themselves, or sourcing the development. All projects state that they currently have no issues with support. Support can be managed in ways they want to. They do report, however, that there may be future problems. Communities are seen as a very important aspect of an open source project, and every project has some form of a community, or wants to set up a community. There is much uncertainty on what a community can do, or how to start a community.

Projects that wish to form a community have the option between three types of communities, each with particular advantages and disadvantages. A *formal community* is a community set up by a central organisation, who actively seeks new members to use the software and join the community. Many projects where a central government organisation writes software for lower government use this form. The community is easy to manage and set up, but the whole community is very dependent on the leading organisation. The *informal community* looks like the formal community, but lacks such a central organisation actively recruiting new members. The community has leaders, but this leadership and the whole organisation is based on self organisation by the members. The community is flexible and can grow to the most optimal form, but everything is heavily based on voluntary action. This can be a risk when too few members have the necessary knowledge,

and others have no incentive to get this knowledge. Finally, there is the *gated community*. This is a protected form of a community, where only members of the community get access to the source. By asking a fee for being a member, support is guaranteed by the community. While such a construction is not open source according to the open source definition, many of the advantages of open source software are preserved.

While support is currently dealt with sufficiently, much can be improved when it comes to forming communities. The users of the project, not some distant organisations, should form the community. Communities should be self organising, determining their own growth. Central government should only create an environment where open source software and communities are possible. Another important aspect that should be improved is information about open source software. On several levels knowledge should be increased. First, users in government organisations should know they use open source software, and why this software is used by their organisation. This creates a better open source culture, where people are more enthusiastic about open source software. And second, on the level of administrators and managers knowledge should be improved. Many administrators do not know exactly what software other government organisations are using. Valuable open source solutions can be missed by organisations looking for a software product. Also, there is uncertainty about how to run an open source project, or how to set up a community. Project leaders of other government open source projects can help here. A central mailing list would provide a place for asking questions about very diverse subjects of open source software, and such a list could create valuable contacts between government organisations.

Table of Contents

Illustration Index

Table Index

Introduction

In any professional organisation software has a very important place. Government organisations are no exception to this rule. Software is important, and with integration of more processes software is still gaining in importance. The software itself should be good. It should do the tasks it was designed for. But software must also be managed. Support on software – in essence keeping the software or the system running – is of the utmost importance.

Software itself comes in many shapes and sizes. Some applications are very small and simple, others are complex systems with much integrated functionality. Software can be made by commercial software vendors, or it can be made by the organisation who uses the software. The software can be general software, useful for many, or very specific to just one organisation. Finally, some software is very expensive, while other software is free.

A special type of software is getting more and more attention lately: open source software. This software is free to use, and free to adapt. Government organisations view this as highly wanted characteristics and show interest in this software. Government organisations are free to select the software, use it, adapt it, and redistribute it. For the central government, broad use of open source software can stimulate innovation, the local economy, and reduce market power of the current software companies.

But such freedom also has its problems: support can be complicated. If one company writes and sells the software, this company can also deliver support on the software. For an organisation it is clear who to turn to in case of problems with the software. But once everyone can use, adapt, and redistribute software, support is not so simple.

This report is written in cooperation with OSOSS, the Dutch government organisation for open source software. OSOSS gives information on open source software, helps other government organisations to give open source software a place in their IT strategy, and brings organisations together so they can learn from each other.

Many government organisations want to use open source software, but there are many questions on how support can be dealt with. This report is written to answer these questions. The research goal is to provide insight in, and give recommendation to improve support on open source software in (Dutch) government organisations. To fulfil this goal, the following question is posted as the central research question:

In what different roles are government organisations involved in open source software development, how do they deal with open source software development, and how can support for this software best be organised?

This questions consists of three parts. First, the role of a government organisation is important. A government organisation could only use the software, provide feedback on development, do some small development, or all development. Especially development is an important concept, one to look into further. Hence the second part of the question. Once it is known how governments (want to) use open source software, the question of how to organise support can be answered.

Research questions

In this report, three concepts are central: support, open source software, and government organisations. For these concepts there are a number of corresponding sub questions:

- *How is support on software generally organised?*
- *What is open source software, and what is the exact role of a community in an open source project?*
- *Why do government organisations use open source software?*
- *What role can a community play in the support on an open source software product?*

These sub questions can all be answered using a literature study. This provides a solid theoretical background for the remainder of the report. That part is more practical in nature. To research how support can best be organised, it is important to know something about current practices in government organisations when it comes to open source software. To study this, a field study is conducted. Using the results of this field study, combined with the theory, the following sub questions will be answered:

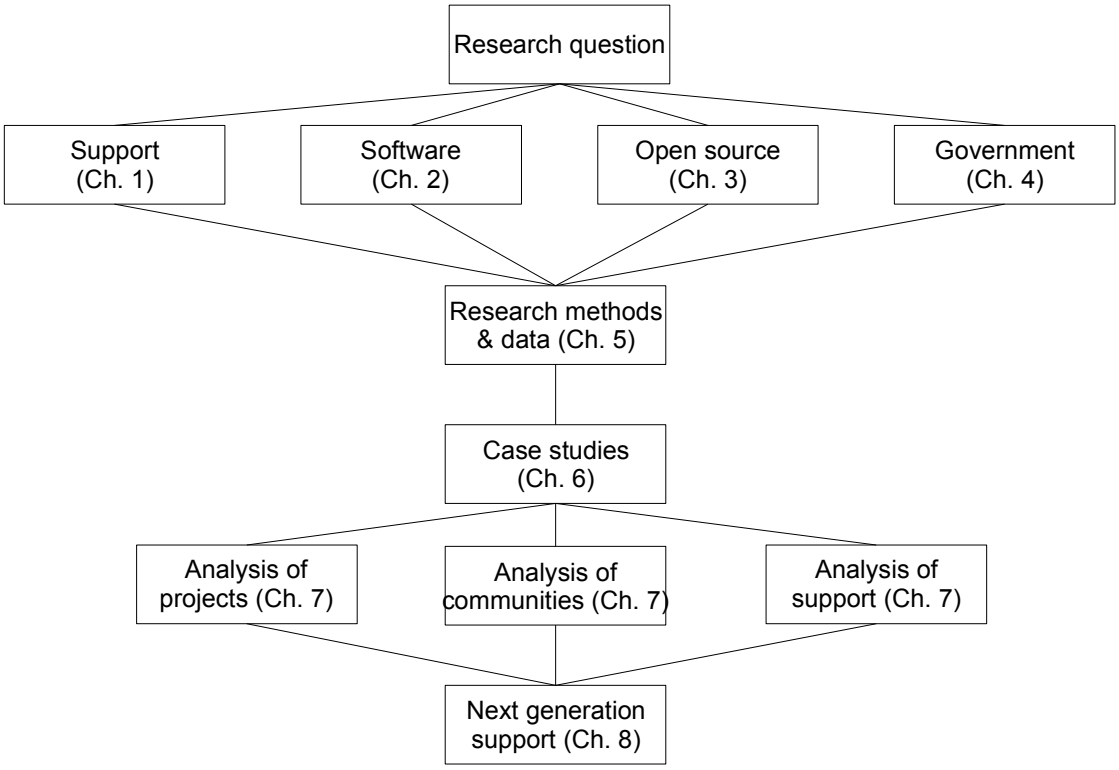
- *How are open source projects of the government organised? Are there any differences with traditional open source projects?*
- *How are open source projects of the government supported? What is the difference in support with closed source software?*
- *What different forms of communities are there in a government environment? Do these communities differ from traditional open source communities?*
- *Can communities play an important role in support on government open source projects? If so, is it needed to stimulate communities? And what is needed to stimulate these communities?*

Report outline

The figure on the following page details the outline of the report. In this introductory chapter the research question, and sub questions are posted. Four fields of literature are important: support, software, open source software, and the government. Each of these fields will have a chapter. The central question of this report is concerned with support. Therefore, Chapter 1 is dedicated to software support in general and theories about software support.

The main focus of this report is on open source software. But before we can turn to open source software, some things about software in general have to be studied. This is done in Chapter 2. Chapter 2 will introduce open source software, and this will be further studied Chapter 3. All different aspects of open source software will be dealt with, from the definition of open source software, to how the software is usually done, and the communities that form around open source software. To better understand support on open source software, an extension to the framework for support presented in Chapter 1 will be introduced near the end of Chapter 3.

The last theoretical chapter is about reasons the government has to use open source software. The premise of this report is that using open source software creates support issues. Therefore it is important to understand why the government still wants to use open source software.



Using the theories a number of the sub questions can then be answered. This leaves a number of questions that must be answered by other means. What questions that are is discussed in Chapter 5. This chapter also discusses how the remainder of the questions will be answered, and how the theories from the first four chapters play a role in answering them.

The main method of research is case studies. The case studies are presented in Chapter 6. Here, a description of all the cases will be given. Chapter 7 is dedicated to the analysis of the results from the case studies. This analysis will be done in three parts. First, the projects themselves will be analysed. From the theoretical chapters it will be made clear that communities can play an important role in the support of open source projects. Therefore, these will be analysed also. After these are analysed, support itself can be analysed. Chapter 7 will make clear that many things on support can be improved. Especially communities can play an important role in the future. Chapter 8 will look into possible improvements. Chapter 9 will conclude this report, by answering the research question, and giving the conclusions of the report, along with the recommendations.

Chapter 1 Software support

Within any organisation support for software is very important. Without adequate support many organisations will not consider using a software product (Phillips, 2004). In this report, support will be defined using a custom definition:

All actions needed to continue the use of a software product within an organisation.

This is a very broad definition. It includes many different actions, ranging from answering questions from users to development of the software itself. In the literature three different forms of support are identified: functional management, technical management, and application management (Van der Pols, 2001). *Functional management* is concerned with keeping the functionality of the IT resources up and running. The team concerned with functional management is the owner and initiator of IT services in an organisation. Functional management is mostly concerned with the users of an IT system. *Technical management* is concerned with the computer systems themselves. That is, servers, operating systems, database software etc. When one of these does not work it is the responsibility of the technical management team to fix it. *Application management* is concerned with specific applications running on the computer systems administered by the technical management team. To keep these applications running adjustments have to be made when needed. Phillips (2004) calls this 'software maintenance'. Three broad categories of software maintenance can be defined: adaptive, perfective, and corrective. Emergency could be identified as a forth category, or it can be placed under corrective actions. Adaptive actions are actions needed to adapt the software to a new situation. This can either be a new software platform (e.g. a new operating system installed by technical management), or changes in the real world, for example changes in regulations that require the software to function differently (e.g. store more information). Perfective actions are actions needed to perfect the software. An example would be to make the software quicker, or easier to use. Corrective actions are actions needed to fix bugs in the software. Emergency actions are actions needed to fix a bug immediately. This is needed when there is a security risk, or when a critical application suddenly breaks. Emergency actions are often 'quick-and-dirty' changes to fix the problem immediately. Corrective actions are more thought through.

For the different forms of support frameworks were developed to better understand the processes and actions involved. For application management there is the Application Services Library (ASL) framework. Because this report focusses on individual applications, the ASL framework is the most important framework to analyse.

1.1 ASL framework

When software is implemented in an organisation, it is the responsibility of the IT department to keep it running. To understand better the actions needed for application management the ASL framework can be used by an organisation. This framework will be described next.

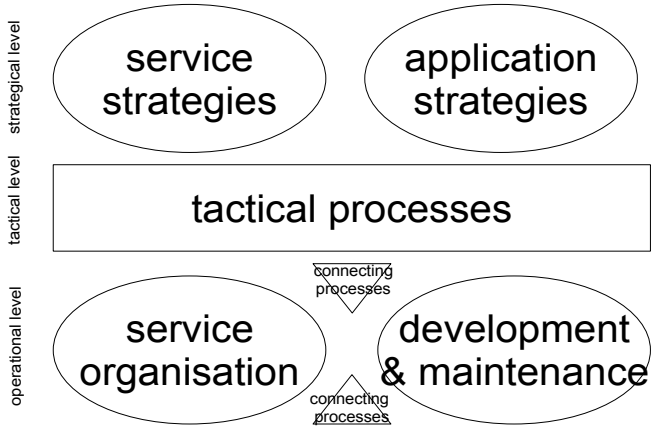


Figure 1: The ASL framework (Modified from: Van der Pols, 2001)

The ASL framework is divided into three layers (see Figure 1), the strategical level, the tactical level, and the operational level (Van der Pols, 2001). Two of these levels are further divided into processes. In the complete ASL framework each of these processes are again divided into specific tasks. When relevant these individual tasks will also be described.

The first part of the strategical level is *service strategies*. In the complete framework this is called organisation cycle management. This consists of processes for developing a vision for IT services. Here, the vision is not focussed on exactly what application to use, but rather what services to offer that need an application. Another important action here is to keep an organisation flexible, not dependent on others. In essence it is a task to prevent vendor lock-in.

The second part of the top layer is *application strategies*, or application cycle management. Where organisation cycle management is concerned with a vision for the organisation, application cycle management is concerned with a vision on what kind of software to use, when to use it or when to implement it, or when to retire an application. The service strategies and application strategies both make up the strategical level of the framework.

The second layer of the ASL framework is the tactical level. This layer can be divided into four separate processes: planning and control, cost management, quality management, and service level management. *Planning and control* is concerned with management of time and labour. Working in an application management team can be quite busy, especially when many things need to be done at almost the same time. Planning and control can help structure tasks. *Cost management* is related to the costs of the services offered. Good cost management helps to make proper cost-benefit decisions for the organisation. *Quality management* is concerned with monitoring the quality of the products or services offered to the organisation. Finally, *service level management* is concerned with setting a desired level of service management, monitoring this level and making adjustments when needed.

The last layer of the framework is the operational level. This layer is divided into several processes, each again divided further. The first part of this level is *service organisation*. The goal of service organisation is to make applications function as best as possible. Actions that belong to service organisation are incident management, configuration management, availability management, capacity management, and continuity management.

The second part of the tactical level is d*evelopment and maintenance*. Not all applications will work flawless. Changes need to be made to the application. These changes are often the result of requests from users or the management team. When a change is identified its impact must be analysed, and a solution must be designed. When the solution is deemed good enough it can be realised (coded), tested, and, when good, implemented in the application. Service organisation and development and maintenance are connected by *connecting processes*. One is *change management*, delivering change requests from the processes in service organisation to development and maintenance. For example, when a user finds a bug in the software, this is reported to the team. For this the process *incident management* is used. This process can generate a change request, where the change itself is dealt with by the development and maintenance cycle. Once the change has been implemented in the software, the second connecting process, *software control and distribution*, comes into play. The software with the change must be distributed to all users so everyone can enjoy the new change (Derksen and Noordam, 2006).

The complete ASL framework, including all sub processes, is depicted in Figure 2. For this report, it will not be necessary to focus on all sub processes, but for completeness the whole framework is included.

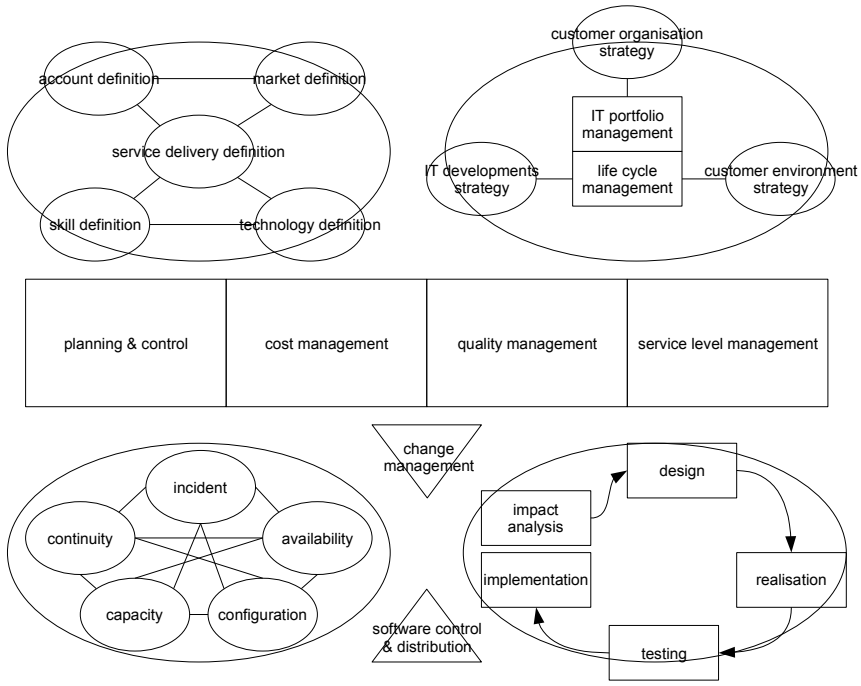


Figure 2: The full ASL framework (Derksen and Noordam, 2006)

The ASL framework describes what actions must be taken by any organisation to have good support on the software they use. On different strategical levels actions must be taken to have good support. Of course the properties of the software also matter: on some software the support will be easier than on others. The type of software is of particular importance. How the software is made and under what licence that software is distributed. This will be dealt with in the next chapter.

1.2 Conclusions

This chapter answered the first of the sub questions of this report: how is support on software generally organised? Before answering this question, the exact meaning of support is defined: all actions needed to continue the use of a software product within an organisation.

For support there are a number of frameworks and models that help better understand the actions needed to guarantee good support. One of such frameworks is the Application Serviced Library (ASL) framework. This framework defines actions of three different levels: the strategical, the tactical, and the operational level. Some of the actions must be conducted by the organisation, while others, such as development and maintenance, can be sourced to specialised companies.

Chapter 2 Types of software

Software does not come out of nowhere, it is programmed by programmers. These programmers use special languages – programming languages – to write their programs. Such a programming language is understandable for humans. Programs are stored as source code. For a computer this source is just text. Before a computer can work with a program written by a programmer, it must be 'translated' to a machine language. In computer science this translation is called 'compiling'. To run a program on a computer the compiled form of the program is needed. To make changes to a program the source of the program is needed. A program can be changed, after which it must be compiled again to make it runnable for a computer. With some software this source code is available, so anyone can make changes. For other software packages only the original programmer has the source code, and only he can make changes to it.

There are different types of software. Of course software can be divided by function (graphical software, office software, etc.). But software can also be divided by other ways. Two ways are of particular concern. First, the division between off-the-shelf software and custom-made software. And secondly the division between closed source software and open source software.

2.1 Off-the-shelf software vs. custom-made software

Off-the-shelf software is software that can be get “off a shelf”, installed and used right away. It is almost always software for general tasks. Examples are operating systems, office software such as word processors or spreadsheets, web browsers, etc. This software can be implemented in an organisation without the need to change anything to the software. As a result many organisations run the same kind of software for common tasks.

Custom-made software is software that is made specifically for an organisation for a specific task. Such software can be either made by the organisation itself, or it can be sourced to a specialised software company. In either case, the resulting software is unique and fully adapted to the organisation's specific wishes.

2.2 Closed source software

Without the source a program cannot be changed. No features can be added, nor can bugs be fixed. Most companies writing software keep the source code secret. Revealing the source code would allow anyone to make changes to the program. Such companies want to keep control over all the versions of their software. If anyone could make changes, many different versions of the same program could be in use at different companies, resulting in difficulties to support or even incompatibilities between different versions. Source code is a creative work, and therefore protected by copyright. These copyrights give companies a means to enforce secrecy of the code. Because the source code is secret, and closed to normal users, it is often called 'closed source'.

Types of software	**Off-the-shelf**	**Custom made**
Closed source	Windows, Photoshop	Custom made inventory system for a company
Open source	Firefox, Apache web server	Content management system based on Typo3

Table 1: Examples of types of software. Any combination is possible

2.3 Open source software

Open source software turns the paradigm of closed source software around. Instead of keeping the source code secret, it is open for all. Everyone can look at the code. Using copyright a special licence is attached to the source code, determining ways the source code can be used. There is a great variety of licences, from absolutely no restrictions to very strict. The Open Source Initiative (OSI) defined open source licenses using ten criteria (OSI, 2007). These criteria will be discussed in the next section. A lot of individuals, companies, and government organisations already use open source software. Famous examples of open source software are *Linux*, a free operating system, and *Mozilla Firefox*, a web browser. Many other programs are also available as open source. Many web sites run the web server *Apache*, use the database *MySQL*, and the scripting engine *PHP*. All these software products are open source.

All combinations of types of software occur. Table 1 lists some example of software products, with their types.

2.3.1 The Open Source Definition

According to the Open Source Definition of OSI "[o]pen source doesn't just mean access to the source code" (OSI, 2007). There is a list of ten criteria to evaluate how open the code is. Only with all ten criteria present the code can be called open source. The first one is (i) *free redistribution*. Anyone should be able to distribute the software, either by selling it or giving it away. However, when a program is distributed it should (ii) *include the source code*, or the source code should be easily accessible from a public place. When source code is modified, and this modified version is distributed it is called a derived work. A true open source licence should (iii) *allow a derived work under the same licence*. However, the original author of a piece of code can state that only the original code may be distributed. In that case the licence must (iv) *allow 'patch files' to be distributed with the source code*. With these patch files a user can apply the changes of somebody else (called patches) to the original source code. Next, the definition lists several criteria to make sure open source software is open for everyone. An open source licence may not discriminate against (v) *persons or groups*, or (vi) *fields of endeavour*. The licence must (vii) *apply to anyone who receives the program*, without the need for additional licences. Also, the licence must (viii) *not be specific to a product*, meaning a piece code may be taken from one program and incorporated into another (providing the other conditions are met). Furthermore, a licence may not (ix) *restrict other software* from being distributed or run alongside it. Last, the licence must be (x) *technology neutral.*

Lerner and Tirole (2002) identify three broad categories of licences: unrestrictive, restrictive, and highly restrictive. These terms apply to how the licence can be carried over to derived works. According to the OSI definition a licence is open source if it allows a derived work under the same licence. However, a licence can also be an open source licence if it allows a derived work to have another licence. One of the unrestrictive licences

identified by Lerner and Tirole is the BSD Licence. This licence allows a derived work to have another licence, as long as credit is given to the original copyright holder. By contrast, the GNU General Public Licence (GPL) is a highly restrictive licence. Derived works may only be published under the GPL. This means that once some code is licensed under the GPL, it will have that licence indefinitely. If a piece of GPL code is incorporated in another program, that program must also be licensed under the GPL. That is why some authors have called the GPL "viral", it spreads itself and 'infects' – that is, places its own licence on – other code. Restrictive licences have some threats of unrestrictive and some of highly restrictive licences. For example, software released under such a licence must remain free, but can be used in commercial products, without the requirement that the total packages must be released under that licence.

The type of licence is important for the incentive of commercial companies to do something with the software. When a software product is licensed under a unrestrictive licence, a commercial company can take the code, modify and improve it, and then sell the improved product. With a highly restrictive licence, the code must remain open, limiting the possibilities of making money from it. The only means to earn a profit from highly restrictive code is to offer complementary services around the software, such as support.

2.3.2 Open source software as a means of practice

In the strictest definition open source software is just defined by a licence. A work is classified as open source if it has a licence that is, according to the definition, an open source licence. But most open source projects also have a very specific way of doing things. It differs greatly from the traditional way commercial companies program their software. A very influential article on the working of open source software is "The Cathedral and the Bazaar" by Eric Raymond. Raymond classified the classic way of programming as cathedral style. According to Raymond, this software is "carefully crafted by individual wizards or small bands of mages working in splendid isolation" (Raymond, 2000). By this he means that software is written by a small number of people, working alone or in small groups. And like the building plans of a cathedral the whole process is planned from the start. There is a blueprint of the design, and the programmers follow this blueprint. After the work is completed as specified in the blueprints, the programmers test it internally. When satisfied, they might release a test version to the public before releasing the final version.

Raymond observed that many open source projects have a very different style, which he named the bazaar style. In this style software design looked more like "a great babbling bazaar of differing agendas and approaches [...] out of which a coherent and stable system could emerge only by a succession of miracles" (Raymond, 2000). In this style there is no careful planning, no blueprints, and no internal testing. In many cases there is not even one company directing it all. Just people doing what they like best: programming. Most pursue their own agendas. They work on a certain product because they like the product, because it suits their needs. When they find a bug in the software, or want an additional feature they program it themselves. This programming is mostly for their own benefit. They do however give their changes to all other users of the product. In return that person also gets all the changes from all other programmers. It is indeed somewhat of a miracle that this way of writing software works at all. Because people pursue their own agendas they write the kind of code they like. They add features they want to see in the product. By keeping in constant communication with each other the development can be coordinated. When programmers and users of the software are in constant contact with each other they form a *community*.

2.3.3 Stages in open source development

Most open source projects start because the initial developers perceive a personal need for it. Raymond (2000) says that "[e]very good work of software starts by scratching a developer's personal itch". The initial developer, the copyright holder, can place any licence on the work, including an open source licence. In licence, the work is now open source, but that does not say anything about the practices. Making a work open source does not imply that new developers immediately begin hacking the code. New developers only get active when they also have a need, or an *itch*, for the project. The new project must hold some promises for new developers (Fogel, 2006). This can be some early version of the program, something to test and play with. Or it may only be a design document, detailing what will be done in the future. Attracting regular users is not one of the first goals of a project. In the early stages of the project there certainly will be many bugs in the program, things that don't work, and features that are missing. Would users start to use a program at this stage, they most likely will be disappointed and turn away from the project. Chances are small that they will ever try the program again.

Once the program releases a first version deemed stable enough for general use it can begin to attract regular users. Before this time the program was only used by the developers themselves or experienced testers. Once the program is officially released it can attract new users. There is no guarantee users will actually use the program. As with the developers, users must have a need for the program. If the need is there, they must also be able to find the program, and be able to use it properly within a reasonable time, or else they will give up and turn away. But if a project is successful it will start to grow slowly. It will attract new users and new developers. Now a true community is forming, one that can continue to develop the program for a long time.

2.4 Conclusions

There are different forms of software. Two important distinctions are by type and by licence. Software can be custom made for a single organisation, or it can be off-the-shelf software. Custom-made software is specially adapted or written for one organisation, and not in use at any other organisation. By contrast, off-the-shelf software is very general software that any organisation can use, and that does not have to be adapted to specific situations.

A second distinction is by licence. Software can be commercial, closed source software. The author of the software controls who used the software and under what conditions. This is very different from open source software. Here, a special open source licence gives much freedom to use, modify, and distribute the software. Because anyone can modify open source software, anyone can make improvements to the software. These improvements can be shared with all other users of the software. Users and developers of the software form a community. This community-style development is characteristic of many open source projects.

Chapter 3 Open source software, communities, and support

Because open source projects are open to everyone, many people can get involved in a project. Some people have a very active role in the project, and can be considered the leaders of the project. Other might develop something now and then, but are not always active. In most projects the vast majority are not even programmers, they are just users of the project. But even between the users there is differentiation: some users might be more active in a project than others. The collection of all these people involved in a project is called the community. Many theories and models exist about online communities. These models not only deal with open source communities, but communities in general. For example, these models include online discussion groups or gamers' communities.

A practical definition for communities is the definition by Preece. Preece (2000) lists the four aspects an online community consists of: people, purpose, policies, and computer systems. This definition can also be applied to open source communities. Each of the four aspects will be described in more detail below.

3.1 Preece's definition of a community

3.1.1 People

Van Wendel de Joode (2005) describes the communities of an open source project as an onion model, as in Figure 3. In the centre are the core members and project leaders. With most open source projects this core group is formed by just a few key persons. These persons often contribute most of the code, and perform the coordination of the project. Around this core group are the developers. These people contribute code on a regular basis, and have in-depth knowledge of the project. Around the developers are the active users. These people are actively involved in the project, test new versions of the software, and submit bug reports, and might, once in a while, perform simple bug fixes. The last group are the regular users. They use the program because it suits their needs. Some might use the software but have no connection with the community, while others might submit bug reports and feature requests. Within most projects all roles are clearly visible. With each layer further from the centre the number of people grows exponentially. People can also move between layers, mostly from the outside to the centre. Most people start as a simple user, testing the software to see if they like it. If they do they might discover things they find missing, or find errors in the program. Some of these users might even be able to fix some of the bugs themselves, or add a feature they really miss. Slowly they get more involved in the project and move to the centre of the model. People can also leave the community. Real world issues might leave no time to work on the project. In successful projects core members can leave; another developers will take it over, becoming a new core member.

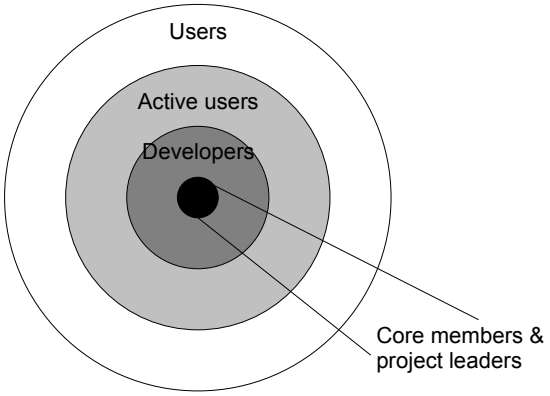


Figure 3: The onion model (Van Wendel de Joode, 2005)

3.1.2 Purpose

The purpose of an open source community is very clear: develop a software product for which there is a need. There are multiple tasks that must be done to make the community as a whole successful in this endeavour. These different tasks are linked to the different roles of people in a community. There are the programmers, who write to code. But besides good code, a community also needs to give support to users of the software. Some instructions on how to set up the software are needed. Answers to questions from users who need help must be given. And users must give feedback to the developers. All these different tasks must be performed by the community as a whole.

3.1.3 Computer systems

Writing software with other people requires coordination. As with all software, people have different demands and want different new features in the program. There will be bugs in the software that need fixing. And then of course there are the egos of the programmers who want their code (or solution to problems) in the product, not the solutions of others. All this requires tools for planning and management. Open source communities have a lot of different tools to aid them. Most projects use most of these tools (Fogel, 2006).

Version control systems (CVS, SubVersion)

Version management for software is very important. With a version management system it is possible for multiple users to work on the project at the same time. All changes they make to the source code are 'integrated' in main source (called the repository). All changes are logged, so changes can easily be undone. It also holds the complete history of all changes, by who they were done and when. It is also possible to make several 'branches' of the code, for example a stable version and development version. The software used most are CVS (Concurrent Version System) and SubVersion .

Mailing lists

In most open source projects e-mail is the most important form of communication between the members. In some open source projects developers have never even met in real life, so all communication goes by e-mail. Special mailing list software archives all mails sent to the list. This makes is also possible to search through past e-mail conversation.

Bug tracker & feature requests

Bug trackers and feature request software is special software designed for these tasks.

Sending bug reports to just one of the developers by e-mail is often not very helpful. Many people receive a lot of mail, and bug reports can easily get lost in all the mails, especially when sent to a busy mailing list. Bug trackers solve this problem. It is a central location to report bugs. The status of that bug is also managed on the bug tracker. When someone is working on the bug, the information on that bug can be updated, and when the bug is fixed the tracker is updated again. Discussion is also possible, for example to discuss proposed fixes. Feature requests work in a similar way, but now only for feature request.

Forums, wiki's & web pages
The web page is often the first contact users have with an open source project. The web pages give an overview of the software. It can also contain some basic documentation about the project. Forums provide a great way to discuss the project and ask questions. In that, the role of a forum is somewhat the same as a mailing list. In many projects however, they are used for a different purpose. The mailing list is more used for technical discussion between the developers. The forums are mostly used by the users to answer questions from other users. Many projects also have wiki's. These look like normal web sites, only users can edit the texts on the pages. Wiki's are often used to build documentation. Because anyone can edit, the documentation constantly improves.

3.1.4 Policies
The last aspect of a community are its policies. These policies are linked to the role of the people in the community. The policies are also enforced by the computer systems used to develop the software. The core developers of the project are often the project leaders. In a community such a figure is highly respected. They dedicate much their time to the software, and try to satisfy the users by constantly improving the software. The project leaders have authority over the project. The most critical part of the open source community is the code itself. Users trust the project leaders to provide good code, and not a program that harms their computer on purpose. There must be control over who can add code to the project, so no malicious code is added. The version control systems take care of that. A password is needed to submit (commit) code. And only people who have proven themselves trustworthy – in producing good quality code – get a password. Other parts of a community are opener. Forums and mailing lists are often available to everyone. Here moderation is needed to enforce policies. Users who break the rules are warned. When they do not behave, they can be banned from the forums or mailing lists. Special software takes care of such bans. Of course such policies cannot truly prevent such abuses.

3.2 Reasons to participate in open source development

Watson *et al.* (2005) define a community as “an organisational form for economic value creation that is characterised by voluntary membership, high autonomy and whose members receive little or no extrinsic rewards”. This definition gives some explanation as to why users would participate in a community, donating their time voluntarily. There are two types of users who participate in open source development: individuals and people from companies. Both can create economic value by participating in the development, but the reasons for participating are different.

Motivation area	Micro level
Economic	Monetary rewards
	Low opportunity costs
	Gaining a reputation among peers
	Gaining future career benefits
Social	Fun to do
	Altruism
	Sense of belonging to a community
	Fight against proprietary software
Technological	Learning
	Contributions and feedback from the community
	Working with bleeding edge technology
	Scratching a personal itch

Table 2: Motivations for individuals to be active in open source communities (Adapted from: Bonaccorsi and Rossi (2003)

3.2.1 Individual reasons to participate

Individuals participate voluntarily. Their costs consist mostly of their own time. But there are many different reasons to participate (Bonaccorsi and Rossi, 2003; Lakhani and Wolf, 2005; Lerner and Tirole, 2000). These reasons can be divided into three broad categories: economic reasons, social reasons, and technological reasons. All reasons are listed in Table 2. Individuals can participate because of (future) monetary rewards. These rewards can come from commercial companies offering money for the first person to program a certain feature. Or it can be future rewards, a good job for example. Because communities are open, they are visible to everyone, including commercial companies. Software companies looking for talented programmers scout the communities. Also, when a company wants something like, or based on, the open source project, the programmers of this project are good candidates for the job: they have the most knowledge about it. Prestige can be an important motivation too.

Social elements too play a role in the reasons to join a community. Many people enjoy programming and join an open source project to fulfil their hobbies. Altruistic elements can also be a reason: people like to give something away. Also the sense of belonging to a community (as in: a group) can be an important social motivation (Britzer, Schrettl, and Schröder, 2004). There are many people who believe in the ideal that software should be free, or the software market should not be dominated by a few large companies. Their reason to join an open source project is to change the market according to their ideas.

Motivation area	Macro level
Economic	Independent of price and licence policies of other companies
	Making money on services
	Selling related product
	Affording innovation
	Hiring good IT specialist
Social	Confirming to the values of the open source community
	Try to sustain cooperation
	Reducing market power of larger companies
Technological	Exploit feedback to cut development costs
	Exploit feedback for testing purposes
	Cut hardware costs
	Promote standardisation
	Addressing security issues

Table 3: Motivations for companies to be active in open source communities (Adapted from: Bonaccorsi and Rossi (2003)

A last broad category of reasons is technological. Participating in an open source project is a great learning experience. This can be of benefit in school or at work. The feedback people get from a community can be a reason to remain active in the community. It can give a sense of being valuable to the community, for example because of expert skills. Many people also like to work with the latest technologies, just because it is possible. A last important reason is that people have a need for it. As Raymond (2000) describes it "[e]very good work of software starts by scratching a developer's personal itch". People have a need that cannot be fulfilled by the current available software and decide to start their own. In this case an open source licence can get more users and developers interested in the project.

3.2.2 Company reasons to participate

Commercial companies run most big open source projects nowadays. For these companies open source software means money. Their primary motivation to join is to earn money or save money. Bonaccorsi and Rossi (2003) have also compiled a comprehensive list of motivations for companies to join open source communities. This is listed in Table 3. Just as for the individual, for companies there are three broad categories: economical, social, and technological reasons. Economical reasons to join a community relate to money. This can either be direct (open source as a means of income), or indirect (open source as a means of cost savings). Companies can sell services around open source products, such as support on software. Or companies hope to sell extra products when their software is free. Cost savings can happen in several different ways. Companies running open source software are less dependent on the developers of a commercial product, and thus have less economic risk. By using open source software it is also possible to lower the costs of innovation. Being active in open source development is, in general, good for a company's image. Therefore, it has a better chance of attracting skilled IT technicians.

Social reasons are important when a company uses open source for its business. Although

no one is obligated to give anything back to the community when using open source software, this is generally expected. Certainly when big software or hardware companies use open source, they have their own developers to aid the project. This of course, is also in their advantage. Having respected programmers in the community gives a company some influence in the direction of the project.

Last there are technological reasons to participate. Companies running an open source project can benefit from all the people in the community. Many community members participate voluntarily. These people give their time to develop and test the program. In a sense a company can get free employees. Other technical reasons are that open source often leads to less hardware upgrades, because an organisation can decide when to upgrade the hardware and software. This process is not dictated by the software developer. Thus it saves costs and network outages. Many open source products follow standards very closely, or even set future standards. When standardisation is an important issue, open source software can be a good choice. A final reason is control. By using open source software, a company knows exactly what a piece of software does. It also knows for sure there are no back doors in the software.

3.3 Characteristics of open source software projects

As identified in the preceding paragraphs, open source software, and open source communities have a number of unique characteristics. Some of these characteristics are unique to open source software, others are just more present in open source than in closed source software. These characteristics will be used extensively in the remainder of this report, when classifying open source projects.

A very important aspect of open source software is its licence. This licence is the first thing that sets it apart from closed source software, whether commercial or freeware. The licence is what allows users to actively contribute to the project, by inspecting the code, submit patches for bugs, or by writing new features.

A second aspect is the community. Around many popular closed source software projects there are also communities. There are many similarities between open source and closed source communities, and even so many differences. Every popular software product is discussed on the internet. Users exchange tips and ideas on how to use the software, and tell what kind of features they want in the next version of the product. Many commercial companies have official communities, but there are also many unofficial communities. Microsoft, as one of the largest closed source software developers, has official news groups where people can post and answer questions. But there are many other news groups, mailing lists and forums where people discuss Microsoft's products. According to Preece's definition all these are online communities (Preece, 2000). But other than communities around closed source software, open source communities play and essential part in the development of the software.

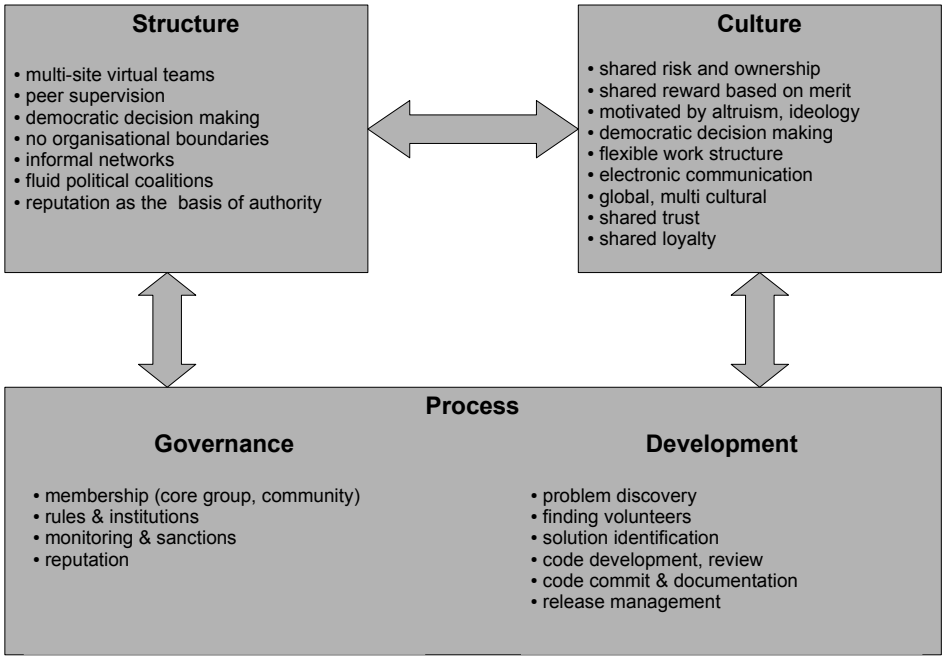


Figure 4: Model for analysing communities (Sharma et al., 2002)

Sharma *et al.* (2002) developed a framework to analyse open source communities based on three dimensions: structure, culture, and process. This framework is based on a framework to analyse organisations. The three dimensions are the same, only the components in each dimension are specific to open source communities. All the three dimensions influence each other.

The complete framework, applied to open source communities, is depicted in Figure 4. The components in the framework imply open source communities are very different from traditional organisations. On the dimension of *structure* the biggest differences are in the way people are organised. There is no strict hierarchy, and no formal organisational structure. This lose organisational structure is reflected in the *culture* of many open source communities. People participate because they enjoy working on open source. Companies participate voluntarily, because they believe it is beneficial to their organisation. Participating organisations adapt to the open source culture in communities, and do not try to impose their own corporate culture on a community. Even many commercial companies starting an open source project seek to have the culture of many existing open source project, and do not apply their corporate culture on their project. This would hinder the involvement of people from outside of the company. Finally, the process of an open source community is very different from that of a commercial organisation. *Process* itself can be divided into *governance* and *development*. Writing code for a commercial program or an open source program is not that different (it is both source code), but the decision making process of what to write is very different. In many commercial companies there is a road map, detailing what to do next. In open source projects such a road map exists, but it is up to the individual programmer to decide what to work on. Governance is also different, not strictly based on authority. Instead, a community has rules. Members of the community

enforce their own rules. People who violate these rules will be addressed. Only in the case of repeated breaking of rules there will be sanctions, such as a ban on using the mailing lists or the forum. Within communities there is some sort of authority, based on reputation. Those who write good code, or provide other valuable services to the community have a higher authority. The community takes seriously the opinion of members with authority, and these people often have a decisive vote in complicated matters.

This framework can be seen as an extension of the definition of Preece for online communities presented earlier. However, this framework has a greater depth. Every factor of Sharma *et al.*'s framework can be mapped to one or more concepts of Preece's definition. This mapping is shown in Table 4. This proves that open source communities are a form of online community as defined by Preece. Because Sharma *et al.*'s framework is much richer in information and just a special case of Preece's general definition for communities, this framework will be used in the rest of this report.

	People	**Purpose**	**Policy**	**Computer Systems**
Multi site virtual teams	X			
Peer supervision			X	
Democratic decision making			X	
No organisational boundaries	X			
Informal networks	X			
Fluid political coalitions	X			
Reputation as the basis of authority			X	
Shared risk and ownership		X	X	
Shared reward based on merit		X	X	
Motivation by altruism, ideology		X		
Democratic decision making			X	
Flexible work structure	X			
Electronic communication				X
Global, multi cultural	X			
Shared trust		X		
Shared loyalty		X		
Membership	X			
Rules & institutions			X	
Monitoring & sanctioning			X	
Reputations			X	
Problem discovery		X	X	
Finding volunteers	X			
Solution identification		X	X	
Code development, review		X	X	X
Code commit & documentation		X	X	X
Release management			X	X

Table 4: Mapping Preece's definition to Sharma *et al*.'s framework

3.4 Support on open source software

Commercial software vendors sell support on their own software. Organisations buy support in the form of service level agreements (SLA), guaranteeing some form of support on the software (Schmitz and Castiaux, 2002). Of course, the more an organisation pays for the SLA, the better the support they will get. If there are questions the commercial company will answer the questions. When there are problems with the software, these problems can be reported to the developers, who in turn will fix the problem.

For open source software the picture is less clear. Many open source licences state that the software comes without warranty. In open source licences this is often described that the software comes "as-is". The developers are not responsible for errors and do not promise to fix any errors. And because the community is made up of different individuals, each with their own agendas, there is no guarantee anyone will answer questions. This however, is the theoretical picture. In practice many developers listen to the wishes of the users, and fix bugs when they are found. Users in the community answer questions from other users. Lakhani and Von Hippel (2001) studied this phenomenon. They call it '"free" user-to-user assistance'. In their study they observed the Apache community. They try to answer the question why users would voluntarily help other users solve their problems. In total five important reasons were found. These same reasons can also be found in the motivations to be active in an open source community. First, users expect reciprocity from helping other; they expect an answer when they have a question. Second, users are "helping the cause". They believe this project, or open source in general, is important and do what they can to help it further. Third, users believe they will gain reputation or will have better job opportunities by answering questions. Forth, answering questions is rewarding. Users might answer questions simply because its fun. A fifth and final reason could be because it is part of the user's job, for example working for a commercial company selling services around open source software. By answering questions correctly the company establishes itself as an expert on the project. Organisations looking for commercial support turn to experts. Thus, answering questions can be a way of marketing (R. van Wendel de Joode, personal interview, 15 January 2007). Another reason is identified by Fogel (2006). When answering questions, developers can encourage users to solve the problem themselves. By doing so they invite people to join the project. Most people will probably decline such an invitation, but a few might actually get involved. Answering questions can be a way of attracting new developers to the project. By answering general questions, developers, and other skilled people in the project, transfer their knowledge to other people, thus increasing the population of potential people to become developers. It is in the interest of the project as a whole to answer questions and help people with their first steps in the project.

3.4.1 Community support

Communities can deliver support (Lakhani and Von Hippel, 2001). New users can ask question using the project's mailing list, a special forum, or via chat. Often projects have web sites with documentation on how to start. With tools such as a bug tracker users can see if they are to only one with a specific problem, or if it is a known bug. The quality of the support differs greatly from project to project. A lot depends on the quality of the community. The quality of a community depends on many variables. These variables are related to the four aspects of a community: people, purpose, policy, and computer systems (Preece, 2000). When looking at people, the first important thing is size. Both the number of users and the number of developers are important. When there are no users, the software is not successful. When there are many users but only one developer the project as a whole

is vulnerable. There should also be diversity among the users. Some users should sometimes act as developers. In general, in every layer of Figure 3 (see page 13) there must be enough users. A project can only be successful when there is a clear purpose. The community as a whole should always be reminded of the purpose of the community. This helps the project to keep its focus. With a clear purpose the community can decide what to work on next, and in what direction to go in the future. Quality of the community also depends on the policies. A project must have some sort of leadership, but people in the community should not get the feeling their leaders behave as dictators. Community leaders should be open to suggestions from people outside the leadership circle (Sadowski and Rasters, 2005). The general management structure is also important. Van Wendel de Joode (2005) concluded that communities are self-organising. A community functions best when it can decide (as a community as a whole) on its own future. A team of leaders or management team can hinder such community decisions. A final aspect for a successful community is the computer systems used. There should be easy to use mailing list, forums, wiki's, etc. so every new user is immediately familiar with such tools and has less problems to use these tools to contact the community. Since many open source community are virtual communities – people often never meet each other – computer systems are the means of communication.

All this leads to an important conclusion: successful communities can give better support than unsuccessful communities can. Studies done by Lakhani and Von Hippel (2001) certainly suggest such a conclusion. Fogel (2006) and Van Wendel de Joode (2005) also support this conclusion.

3.4.2 Commercial support on open source software

All support from the community is informal. A community provides no guarantee that problems will be dealt with. For professional organisations this informal support might not be good enough. And if there is a demand, there will be companies delivering it. There are many commercial companies offering support around an open source product. Most of these companies are specialised in one specific product. They have experience with one product, and often contribute actively to the community. Such companies can help organisations with the implementation of the software, or offer support on the software. However, not all open source software has such commercial support available. The amount of support depends on two things. First, companies will only offer support if there is enough demand. Second, a product must already have a good functioning community (Wernberg-Tougaard et al., 2007). This will guarantee the ongoing development of the product, without the need for the company to do all development itself. With a strong community a company can 'exploit' the community for development and testing of new versions.

Not every open source project will attract a large enough community. A successful community depends – among other things – on critical mass. Enough users must be willing to participate in a project. Without enough users and developers the software will not evolve, and thus will attract even less users. A critical mass is needed to guarantee the ongoing development of the product. The critical mass for different projects is not the same. This is illustrated in Figure 5. Projects that are used by many users (operating systems, office applications, etc.) need a larger number of initial users to achieve a critical mass. These software products are general applications that are only interesting to use when a very large number of users use them. For example, a complete operating system is not interesting to develop when only one user uses it. On the other hand, very specific software can be interesting even when there are very little other users. The critical mass needed to make such applications a success can be lower than for more general

applications. This critical mass is not related to the complexity of the projects, as Figure 5 show. Very complex products that are targeted at a niche market can get enough critical mass to sustain the project.

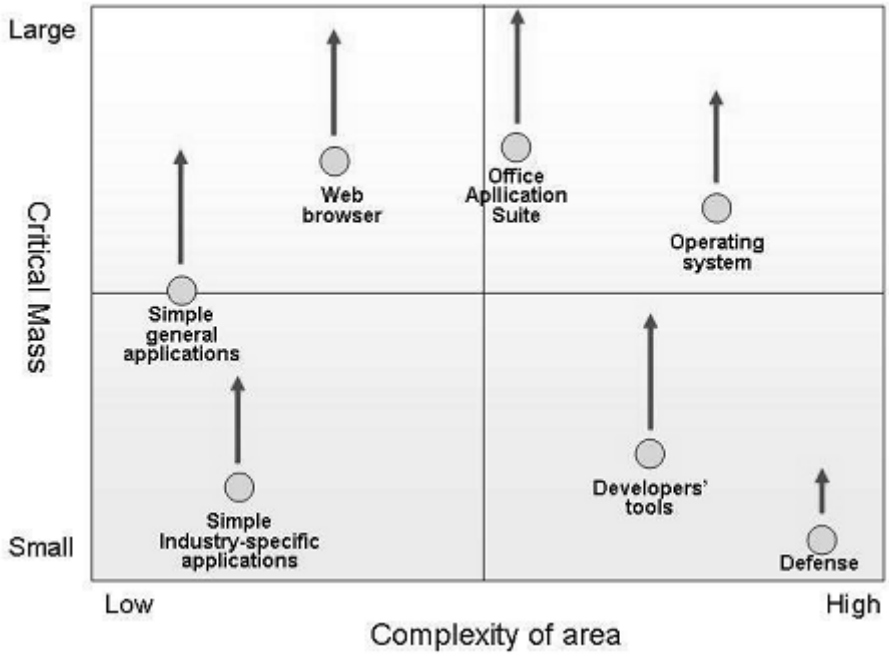


Figure 5: The relationship between the complexity of the area and the critical mass. From the points upwards a project has enough critical mass to support itself (Wernberg-Tougaard, 2006)

When a community has reached a critical mass and has proven to be a success, commercial companies will get interested in the product. This is often because licence costs on these products are zero. Most web hosting companies support Linux and Apache. These products provide the necessary functionality with modest costs. For more specialised applications it can be more difficult to get support. The potential market for these products is not large. Nevertheless, because there are no licensing costs in adopting the software, there is money for buying added services. This makes even smaller open source projects interesting for companies, especially for small and medium sized enterprises.

3.4.3 Support and language issues

A complicating issue for getting support lies in the language. On the internet and in communities English is the most common language. Although many open source products provide features to translate the interface into another language, English is the *lingua franca* for communities. Discussions about support and proposed features are conduced in English for many international projects.

Most Dutch government organisations only want to use software if the interface is in Dutch. Without a good Dutch interface the software is not considered for adoption (Personal interviews; see case studies and Appendix F). This can have two important implications. First, using existing open source software can be troublesome. A good Dutch translation might not be available, thus good quality software might be ignored. When there is a translation, using the translated version can be problematic for a community. It can be awkward to use English to communicate with the community, especially when

referring to certain features using the translated terms. A second problem relates to software started by Dutch government organisations. When writing software, this software will be in Dutch. Possible all documentation will also be in Dutch. While this might be a plus for many Dutch organisations, it is difficult to get a large, active community with only Dutch members. By not providing the interface, documentation, and help in English, the number of possible users is limited. This makes is difficult to get a critical mass. Providing everything in English creates new problems. The software has to be translated (partly) even before there are many international users. And all communications have to be done in English so everyone can read and join, even though the people communicating all speak Dutch.

3.4.4 ASL framework and open source software

Chapter 1 introduced the ASL framework for looking at support in an organisation. The ASL framework was not developed with open source software in mind, but can very well be applied to open source software. To use the ASL framework in the contents of open source projects, the model must be extended. When the ASL framework is applied to open source software, not all processes of the framework remain confined to one organisation. Also, the community becomes a integral part of the framework.

Many of the processes in the ASL framework are the same for open source and closed source software. After all, open source software is just another form of software. An organisation still needs a vision on what processes to support with software, and what software to use for that. It also still needs to manage its costs and labour. Just as closed source software, open source software has to be implemented in the organisation. Support needs to be given to the end users, there are incidents to collect, etc. In that light, open source software is no different from closed source software. However, with *development and maintenance* there are big differences. Other than closed source software, open source software can have a community that does the software development. This community is not some distant entity, but the organisation itself is part of the community. They can have influence on the direction of the project, report bugs to the community, give feedback, etc.

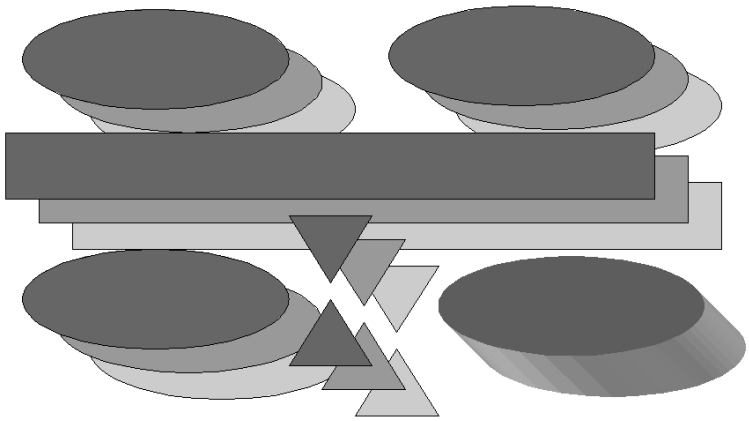


Figure 6: The combined ASL framework of three organisations using the same open source product

Figure 6 tries to represent this situation graphically. Here there are three organisations using some open source software product. Each organisation is responsible for their own strategical (top layer) and tactical processes (middle layer). Parts of the operational processes are also separate for each organisation. However, there is cooperation between all three organisations on development and maintenance (lower right). These processes of each organisation are connected, forming a single set of processes. This is what is called the community. But the community can be more than just the sum of development and maintenance processes of different organisations. Commercial companies can be part of the community, and hobbyist users too. Development and maintenance is now driven by three different inputs. First, the change requests from the organisations in the community. Secondly, changes can be implemented by commercial companies on their own accord. This might be done by a company to improve the software, thus giving the software a better position in the market and creating business opportunities for the company. Finally, hobbyist users can also implement new things, for the reasons listed in Table 2. Figure 7 represents a schematic view of the total community. Organisations can cooperate on most of the processes in *development and maintenance* (See Figure 2 on page 6). The impact analysis must be done by the organisation itself where it involves impact on other software that the organisation uses. However, the impact of a change on the software itself (what are the effects of a change on the other functionalities of the software?) can be done by all community members. Design and realisation are processes that are especially suited for the community. Here the community members can discuss possible solutions, design different implementations, and decide on the final solution. Finally that solution can then be tested. Where it involves testing of the software itself, this can be done by all community members. If the change fixes the problem, without creating new ones, it can be considered a success. However, the new version also needs to be tested in cooperation with all other software the organisation is running. This is an action that must be done by each individual organisation. After all tests are successful, each organisation needs to implement the new version.

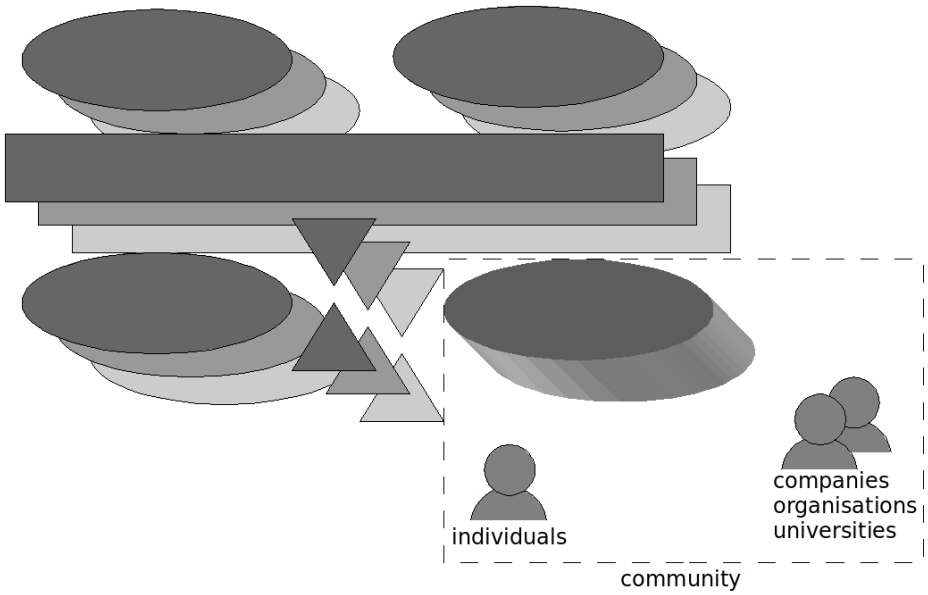


Figure 7: The total community consists of all kinds of organisations using the software and individuals

3.5 Conclusions

Communities play an important role in the development of open source applications. The community around an open source project consists of several different kinds of people. Some people are very active in the project, and program most of the code, while others only use the software. Many more people just use the software, than people that code for it. Graphically this can be represented as an onion, with different layers.

The community is made up of individuals and people working for companies. Both have their reasons to participate in the community. The community as a whole can give support to the other users of the software. This can be in the form of user-to-user support. It is also possible to get commercial support on open source software. A company will provide this support is there is a large enough critical mass, so it can profit of the work done by the community as a whole. From the perspective of organisations using open source software, the community can play an essential support role in the action *development and maintenance* from the ASL framework. Here, organisations using the same software can pull their resources together so support is better for each organisation.

To analyse communities the framework by Sharma *et al*. can be used. This framework treats a community like an organisations, and shows what aspects should be present in good functioning communities.

Chapter 4 Governments and open source software

The previous chapter introduced open source and listed reasons why individuals and organisations participate in open source projects. However, these reasons do not explain why government organisations use open source, and why some government organisations even have a preference for open source software (Evans and Reddy, 2003). This chapter lists the many different reasons, and also lists some much-heard reasons that are not always valid.

4.1 Government policy

In 2002, Dutch Parliament passed a motion – known as the "Motie Vendrik" – in which the entire Parliament asked the Administration to promote open standards and open source software (Vendrik, 2002). According to the motion open standards are a requirement for all government organisations, and open source software should be promoted. By 2006 all government organisation were to use only open standards. In the motion the argument is used that the software market is strongly concentrated, and only a few market parties have some significant market shares. Because of this oligopoly, society would not profit enough from the advantages software has to offer. Not much has come from this motion. In 2007 most government organisations still use closed standards for the most used documents, and use of open source software is increasing, but not in a way the authors of the motion originally hoped (OSOSS, 2007).

The European Union is also active in open source software and licences. The European Union (EU) financed a great number of reports on the benefits of open source software. Partly based on these reports the EU has a preference for open source software, and encourages all member states to use open source software where appropriate. This promotion of open source is one of the tasks of a special programme, the Interoperable Delivery of European eGovernment Services to public Administrations, Business and Citizens (IDABC) programme. IDABC's objectives are listed as:

- *"Encourage Europe's public administrations to consider and assess the most advantageous IT solutions;*
- *Reduce replication costs of eGovernment solutions and help to spread good practice throughout public administrations;*
- *Ensure that the market for IT solutions remains competitive;*
- *Reduce IDABC's own costs in application development and maintenance;*
- *Help ensuring that open source software solutions are able to compete on an equal footing with proprietary solutions." (IDABC, 2007)*

IDABC also developed a special licence for open source software, the EU Public Licence (EUPL). This licence applies to all software developed for the IDABC programme. The EUPL is an open source, restrictive licence. The EUPL is compatible with the most used open source restrictive licence, the GNU General Public Licence (GPL).

The IDABC's objectives show the EU's preference for open source. In the Netherlands, the OSOSS programme has similar objectives. A few of the reasons why governments support open source are clear from IDABC's objectives. The next paragraphs list all the reasons to

use and promote open source in more detail.

4.2 Total cost of ownership

One of the most heard reasons why to use open source is costs savings (Waring and Maddocks, 2005). Open source software is distributed under an open source licence. Organisations using open source software do not have to pay any licence fees for the use of the software. Compared with commercial software this can be a huge cost saving because licences for commercial software can be very expensive. However, licence costs of software are only part of the picture. There are many more costs than just the licensing costs. Training and maintenance are also very important. For this comparison, the total cost of ownership (TCO) is used. This is a calculation that includes all costs to use the software. Use consists of licensing, installation, maintenance, training of end users and support staff, ease of use (productivity), and loss of productivity due to errors in the software (Cybersource, 2004). Whether closed source or open source software has a lower total cost of ownership is a heated debate. These comparisons are mainly conducted to compare the total costs of Microsoft Windows versus Linux. Some studies concluded the TCO of Windows is lower (CCW Research, 2005), other concluded the TCO of Linux is actually lower (Cybersource, 2004). The problem is that many of these studies are sponsored by either Microsoft or a company having strong interests in open source software. And although all studies claim to be independent, much depends on what is taken into account and what time frame is studied. This makes TCO a fuzzy definition (Knubben, 2004).

4.3 Benefits of open licences

4.3.1 Vendor lock-in

The fact that software is distributed under an open source licence has some major benefits. One of the most important reasons is that open source software can help prevent vendor lock-in (Simon, 2005). When an organisation gets all software and support from one company, that organisation can become very dependent on that company. The organisation is dependent on one software product to access its data. In some ways the commercial company can dictate what other software the organisation must use. For example, when a software vendor releases a new version of a software product that will only run on a newer version of an operating system, the organisation also has to upgrade the operating system if it wants to use the new version. Being dependent on one vendor can also be very costly. When the vendor decides to increase the licensing costs the organisation is forced to pay. There is no alternative to the software, and much of the daily operations depend on the software. This makes strategic planning very difficult. Another danger of vendor lock-in is a bankruptcy of the vendor. The organisation needs the vendor to support the software, but when the vendor goes bankrupt there is no other party to turn to. A solution to this is to use an escrow. This is a legal agreement where the source code is given to a third party, and is only given to the user if the original vendor can no longer work on the code (Kay, 1992). While such agreements can help minimise the chances of a vendor lock-in, using open source licences simplifies these agreements considerably.

Open source software can help to prevent many of these vendor lock-in scenarios. The vendor of the software cannot dictate what other software the organisation must use, nor can it increase support costs. When the organisation is unsatisfied with the performance of the vendor it has the option to turn to another company. The software does not belong to

one company, so other companies can give support. In the event no other partner can be found to support the software, a last option is that the organisation does their own support. This might imply a larger IT department, and this is certainly not the most ideal solution. But when the software is critical, such a thing might be needed.

4.3.2 Code sharing

Using open source software also stimulates code sharing. When one organisation writes code, that code can also be beneficial to another organisation. Open source licences make sharing of code easy. While the same can be accomplished by closed licences (by signing a deal to distribute the code), open source licences provide an easy way of sharing without risks. Open source licences state the code is provided "as-is", so no support has to be given. The open source licence comes in place of formal contracts to share code, or share code by no contract or licence at all.

4.3.3 Customisability, interoperability, and scalability

Open source licences allow the code to be adapted. This way an organisation can write a piece of software that is tailored to a specific situation. The organisation can have the benefits of a customised program, without much of the costs associated with customised programs. That is because large parts of the program are based on something pre-existing. Interoperability is related to customisability. Specific changes can be made to open source software to make it work with other software in use at the organisation. Finally, scalability means that the software can be extended to have more functionality. While all these options are also possible with custom closed source software, open source software makes these options available at a fraction of the costs (Waring and Maddocks, 2005).

4.3.4 Know what the software does

Some organisations within the government work with information that should be kept confidential (e.g. the tax bureau or defence). This means organisations must know for sure that the software they use does exactly what they think it does, and nothing more. A concern are "back doors" in the program, ways by which someone from outside can get access to the data. Because closed source software is sort of a black box, the existence of back doors can be hard to proof. When the licence is open, concerned parties can inspect the code for malicious code.

For many government organisations this is a real concern. For example, Microsoft allows some government organisations to inspect the code of the Windows operating system under a special shared code licence. Certain people within the government may review the code, but the code must remain secret to others. Government organisations have to request such a review, and Microsoft has to approve the request (Microsoft, 2005). Open source licences make this inspection a lot easier.

4.4 Benefits from a community style of development

4.4.1 Reliability and security

One of the most famous lines from "The Cathedral and the Bazaar" by Raymond (2000) is "given enough eyeballs all bugs are shallow". All code contains errors. But when more people look at the code, the chance that someone spots an error increases. So, when many people from the community look at the code the potential number of errors in the code are lower. A conclusion is that, because many people look at the code, open source software is actually more reliable and more secure than closed source software (Paulson, Succi, and Eberlein, 2004). However, this argument is not universally accepted. Many studies concluded that open source software is not more secure than close source software

(Microsoft, 2007). The debate on security and open source software shows some of the same characteristics as the debate on total cost of ownership.

4.5 Spillover effect from open source development

4.5.1 Giving things back to the tax payer

Besides some of the organisational or technical benefits to open source software, there are other reasons why governments might want to use open source software, or release its own software under an open source licence. The government operates from taxes paid by citizens and companies. Therefore, the government has a responsibility to spend this money wisely (Applewhite, 2003). Sharing code is one of the means to do that, since it prevents duplication of software. Another government organisation will not spent money on software if such software is already available at another organisation. Another reason is more ideological: because citizens paid for the software (via taxes) the software should be given back to the citizens. Other individuals or companies might benefit from the code, so there is no reason to keep the code secret.

4.5.2 Stimulate the local economy and improve competition

Software licences cost a lot of money. Much of the software is made by large software companies. Viewed from the position of the Dutch government, many of these companies are foreign. By paying the licence costs money flows abroad. It can be good for the local economy if (part of) that money is spent locally. Open source software makes this possible. Smaller, local, software companies can customise an existing open source application. In the same manner open source software can also improve competition on the software market. In some markets for software there are monopolies or oligopolies. In a free economy it is the government's job to ensure fair competition and to make sure there are no market disruptions. By choosing new open source applications (and not the incumbent closed source applications) the government helps create a level playing field for new or smaller software companies. This way market failures can be corrected (Hahn, 2002).

4.5.3 Promote innovation

An often-heard argument is that open source promotes innovation. Metiu and Kogut (2001) note: “[t]he open development model opens up the ability to contribute to innovation on a global basis. It recognises that the distribution of natural intelligence does not correspond to the monopolisation of innovation by the richest firms or richest countries.” (p. 32) With an open development model everyone can contribute to the code, and every contribution can become something very innovative. Innovative ideas are no longer limited to the employees of a software company. Another reason stems from the saying “necessity is the mother or all innovation”. Users have a need for a specific function but cannot find such a software product in the market. As a consequence they write it themselves.

The idea that open source software stimulates innovation is not universally accepted. Some actually propose the opposite: open source software and licences hinders innovation. Open source developers look at features in commercial products and copy these features in their open source product. Open source is then all imitation, not innovation. In this view constant innovation from open source would even be impossible. Since open source is free there is no incentive to innovate (Forbes, 2005).

4.6 Off-the-shelf and custom development

Clearly, open source software has a number of advantages for government organisations. But that leaves the question what kind of open source software should or could be used: off-the-shelf software (general software) or custom-made software. Open source software exists for any of these situations. Much of the well-known open source software is general software – operating systems, web browsers, web servers, office applications. But sometimes an existing open source package is not completely suited for an organisation. That organisation has then the option to adapt the program. Organisations can also write their own software. The advantage is that the software is fully adapted to the specific situation, the downside is that it costs time and money to write own software. Custom development (both adapting and writing new) can either be conducted in house, or sourced to an external software company.

Custom open source software is best suited to a situation, and holds the most benefits from open source development. When organisations write their own software, that code can be shared, it can be given back to tax payers, the local economy receives the most stimulation (more than by using off-the-shelf software), and it is best suited to promote innovation. But that raises the question whether or not government organisation should start writing their own software. In a study of open source possibilities for the UK government, Waring and Maddocks (2005) write:

"The big question is how do you reverse this trend [of outsourcing] without growing the large Government IT departments of the 1980s? This is a very difficult question and one that has no immediate answer. The Government of the UK would need to overhaul its national IT strategy and pull back from some of its outsourcing deals [...]. This could take years and would mean a huge investment in a new IT infrastructure. [...] [open source software] implementation will only take place where there is a degree of autonomy and where the IT skills are readily available." (p. 423)

Writing custom software takes time and requires enough skilled people on the project. Not every government organisation would have the time and the resources – both in money and employees – to start an open source project. Waring and Maddocks also note that employees must have some degree of autonomy and IT skill must be available. This is directly related to Raymond's "itch" (Raymond, 2000). Employees will start a project when they perceive a need in their organisation. For this to happen there must of course be the freedom to experiment with a new project. Having good IT skills certainly helps to get projects quicker off the ground.

4.7 Government promotion of open source

Another important question is whether the government should actively promote open source software, both the use of, and the development of it. Hahn (2002) notes:

"[S]ome governments have expressly tilted the playing field toward open source software, subsidising its production and use. Instead of choosing software based on its merits – reliability, security, ease of use, and so on – these governments favour open source as a matter of policy. For instance, Singapore is offering tax breaks to companies that use the open source Linux operating system instead of commercial alternatives like Windows. [...] In addition to the handful of measures that have already passed in Brazil, Germany, and Singapore, many more governments worldwide have open source proposals pending, and the European Parliament has called on member nations to promote the use of open source

software whenever practical." (p. 4)

Is such an intervention justified? In liberal economic theories – and Dutch government's economic policies are largely based on these theories – intervention is only allowed when there are market failures, and such an intervention significantly improves the situation. In all other instances the decision for open source software should be left to the market. Open source can have clear advantages over closed source software. However, if there are significant market failures governments might not, or cannot choose open source software, even if they want to.

In Hahn (2002), four different authors list their view on intervention in the software market. Out of these four only Lessig (2002) identifies some market failures. Lessig compares software to public goods – goods that are nonrivalrous and nonexcludable. Licences are used to makes software into a private good. Lessig concludes:

"Between two systems for producing a public good, one that releases the information produced by that good freely and one that does not, all things being equal, public policy should favour free access. This is not because of some egalitarian bias or because of ideals about social equality but for purely neoclassical economic reasons: free access brings the cost of information down to its marginal cost, and neoclassical economics favours price at marginal costs." (p. 59)

Lessig uses an argument for government intervention not based on the properties of the software market, or advantages of open source, but one purely based on economic theory. In that he is the only author in Hahn (2002) who supports direct government intervention. All other authors agree that government organisations should base their decisions for open source or closed source software on the same arguments as profit-maximising firms.

In the articles, one other interesting point is mentioned. Two authors strongly oppose the use of the GNU General Public Licence (GPL) by the government. As discussed in Chapter 2 GPL licensed code has a "viral" character. Releasing code under the GPL prevents the code from being incorporated into commercial software. When modifications to the software must be released under the same licence, companies do not have enough incentive to adapt the software. In their view the GPL hinders innovation. When governments release code because it was paid from the tax payer's money, it should not exclude companies who want to make profit. The discussion about the "viral" nature of the GPL is less strong in Europe (the articles in Hahn (2002) are written from an American perspective). The EU, after all, has its own open source licence, the EUPL, a licence that has the same "viral" character as the GPL.

There is evidence that there are market failures within the Dutch software market. The market for local governments is dominated by just two parties, making it a oligopoly (Baarsma, 2004). In itself this is already a market failure. These two companies force local governments to make certain strategic decisions. All software these companies sell only runs on the Windows platform, and links with Microsoft Office. While there is a demand for software that runs on other operating systems, or can interface with other office applications, these companies only give support on a select number of software packages. According to Baarsma, this is clearly a market failure, and by supporting open source software the government can help solve this market failure. The government can give other companies a chance on this market. If successful the incumbent companies will follow and offer other solutions than the ones offered now.

4.8 Conclusions

For government organisations there are a number of important reasons why they would want to use open source software. Because open source software has no licence costs, costs can be lower. However, much of the costs of software are not in licences, but in training, modifications, support, etc. But in these fields there are large benefits to open source software too. Because no company owns the software, an organisation is less reliant on one single company. Open source can help prevent vendor lock-ins. Because open source software may be studied, modified, and shared, government organisations can easily share software with other organisations, adapt it to the other software they use, and have less security risks because it can be studied what the software does exactly.

The use of open source software can also have a more ideological reasons. Because the government is paid for by the tax payers, it software truly belongs to the tax payers. By releasing it under an open source licence, all tax payers can profit from it. Using open source software can also stimulate the local economy and promote innovation. Support and modifications can be done by local companies, and the community style of development can ensure that anyone with good, innovative ideas can see their idea implemented in the software.

Chapter 5 Research methods and data

5.1 Research questions

In the introduction the main research question and the sub questions were already stated. The main research question is:

In what different roles are government organisations involved in open source software development, how do they deal with open source software development, and how can support for this software best be organised?

The main research question is divided into smaller sub questions:

- *How is support on software generally organised?*
- *What is open source software, and what is the exact role of a community in an open source project?*
- *Why do government organisations use open source software?*
- *What role can a community play in the support on an open source software product?*
- *How are open source projects of the government organised? Are there any differences with traditional open source projects?*
- *How are open source projects of the government supported? What is the difference in support with closed source software?*
- *What different forms of communities are there in a government environment? Do these communities differ from traditional open source communities?*
- *Can communities play an important role in support of government open source projects? If so, is it needed to stimulate communities? And what is needed to stimulate these communities?*

Based on the preceding chapters, some of these questions can already be answered. Government organisations use open source software because using this type of software can have a number of benefits. These benefits lie in less dependence on vendors, better integration with software and hardware already in use, stimulation of the local economy, promote innovation, and in some cases lower (direct) costs. These reasons are convincing enough for a large number of government organisations to start implementing or producing open source software.

As with any software package, support on open source software must be good. The ASL framework provides a clear reference on how support should be organised by an organisation. The standard ASL framework provided details for a single organisation to organise support. However, this framework was not intended for open source software that is shared between multiple organisations. Therefore, an extension of the ASL model is proposed. In this model, several organisations can cooperate on maintenance and development of the software. In cooperating together, they form a community.

The thing that set open source really apart from closed source software are the existence of these communities. Communities provide a vital role in existing open source projects. Here, communities take care of maintenance and development of the software, and of user-

to-user assistance. Whether or not such communities are also present between government organisations remains to be seen. And if communities are indeed present, can they provide the same services as communities from known open source projects?

5.2 Questions left to answer

The first four sub questions could be answered by a literature study. The last four sub question can not. Therefore, other means must be used.

How traditional communities work is explained in great detail in the previous chapters. Also, how support is dealt with on current closed source applications is known: the vendor takes care of development and maintenance, while other actions from the ASL framework must be done by the organisation using the software. The big question is: how do government organisations use open source software, and how is support organised. To answer these question, a field study has to be conducted. Within the Dutch government there are not that many cases that can be studied. For this study only nine cases were found. These cases represent a majority of all projects that conform to the boundaries of the research. These cases, and how these cases will be analysed, will be dealt with shortly. First, the boundaries of the research will be explained.

5.3 Methodology

5.3.1 Boundaries of the research

Government organisations can use open source software in several different ways. Organisations can use large, existing open source applications by simply installing them, or by buying services that include these applications from a commercial company. For these well-known applications, there are enough commercial companies that can give support. Examples of such applications are Linux, PHP, MySQL, or Firefox. For these applications there is, in general, no need for a government organisation to adapt them to their situation. These applications are very so well supported by their community and commercial companies, that the question of how to deal with support is almost a non-issue. For these applications, a government organisation can turn to enough companies for their support needs. When not satisfied with their current support partner, there are enough other companies to turn to. Therefore, such use of open source software is not further studied. It is, however, an important conclusion that support on these software applications is very well possible.

Once a government organisation adapts software, or releases own software under an open source licence – and other organisations start to use that – support problems can emerge. Therefore, only such cases are included in this study. An organisations has to contribute actively to open source development and maintenance, or have this development and maintenance sourced to a commercial company.

5.3.2 Research framework

To study how government organisations deal with support on open source software, case studies were conducted. For this study a total of nine case were selected. These cases come from different government organisations: some central organisations while others are local. Most of these cases are projects where the software has an open source licence. With some of the cases, project have a different licence. In that case the project makes use of open source software, adheres to the philosophy of open source development, and shares most of the advantages of open source software. While not open source in licence, these project

were selected because they can give valuable insight in what different forms there are to use and share software with other government organisations, and how such projects deal with support.

5.3.3 Selection of the cases

All case studies are projects known to OSOSS. The cases studies for this report are brought forward by OSOSS. Some projects were known directly, while others were found by sending a request by e-mail to all contacts of OSOSS. A short description of all the nine projects is listed below. In the next chapter a more detailed description of each project will be given.

Flamingo
A map viewer used by most provinces in the Netherlands. Originally developed as a viewer for risk maps. The software is completely written from scratch.

Watlas
Another map viewer used to present maps of the Waddenzee and surrounding lands. The map viewer is based on the open source MapServer of the University of Minnesota, and combined with other open source tools, such as the Typo3 content management system.

A-select / DigiD
A-select provides a means of secure authentication, using other security measures besides user name and password. DigiD is a partly a fork (code split from the original code) of A-select, partly only some modifications. DigiD is used as the main authentication mechanism for many government web sites.

eFormulieren
eFormulieren is based on the open source development framework ePlatform, developed by Logica CMG. It intended for making online forms easy.

MMProject
MMProject is an extension to EGEMProject, which in turn is based on the open source content management system MMBase. MMProject is a project management tool.

GemGids
A geo information system to view information about a local community in a geographical representation.

Modernisering GBA
A project to implement a new system for the municipal population register for all local governments in the Netherlands.

GovUnited 'Op Afspraak'
A system for planning appointments with local government's civil servants, e.g. to apply for a passport.

APLAWS+
A content management system especially designed for local governments. This is the only non-Dutch project, APLAWS+ originated in the UK.

5.3.4 Case methodology

To study each project, two different data sources were used. First, most of the project have an internet page where they publish information. Especially if the project is very open, this provides a valuable source of data on how active to project is, who contributes to the project, how many people ask questions about the software, etc. Second, interviews were held with the key person from each project. Each interviews lasted about one hour. These

interviews were conducted as topic interviews (Baarda, De Goede, and Teunissen, 2005). The interviews were quite open, with no predefined question, only predefined topics to talk about. This allowed for a greater freedom for the interviewees to talk about all the relevant aspects of their project. For the interviews an interview protocol was made to structure the interview, and make sure all relevant topics were covered. This protocol is added to this report in Appendix B.

After all the interviews, the results were analysed. This happened on two levels. The first level is by looking at common themes in all the projects and comparing them to the theories presented in the earlier chapters. Using the ASL framework support on the projects can be analysed, and it can be tested whether or not communities play an important part in support for government open source projects. All the projects are also compared against the open source definition, and the different frameworks for open source projects. Projects can be viewed in the light of Preece's definition for communities, the Union model for communities, and Sharma *et al*.'s framework for communities. All these framework can show if and how communities compare to open source communities outside the government.

This analysis can give information about how projects are organised, how support is organised, and how the communities function. If communities function differently from communities as detailed in the literature, Sharma *et al*.'s framework can be used to study these communities in greater detail. The framework presents 26 components that should be present in each community to function optimally. Each of the studies communities can be analysed using these components. The exact definition of each of the components is listed in Appendix A. Each of the components is treated as dichotomous. If the component is observed in a project's community, the value is TRUE (1,00), otherwise the value is FALSE (,00). This results in the dataset shown in Appendix C. The observed components for each case are also listed after each case description in the next chapter. Using a quantitative approach, all the communities can be analysed. If components are missing from all project's communities, it will provide clues for strengthening the communities.

5.3.5 Methodology on a question-by-question basis

To explain how each of the sub questions can be answered using the methodology described above, all remaining sub questions are stated again. This will also explain the relevance of answering each sub question for answering the main research question.

> *How are open source projects of the government organised? Are there any differences with traditional open source projects?*

It is very likely that the project studied will not be a homogeneous group. Project will differ greatly in their targets, their scope, and functionality. By studying all of the projects patterns emerge. Some things in government open source projects are the same, while other aspects differ greatly per project. By comparing the studies projects with projects known from literature a clear picture emerges of how projects are organised. Studying the variety of projects will answer the different roles government organisations can have in open source projects.

> *How are open source projects of the government supported? What is the difference in support with closed source software?*

After studying the organisation of each project in the previous question, answering how each project deals with support is a lot easier. Support is related to how the project is organised. By comparing each project's support action with the actions from the ASL framework, it is discovered how good projects can handle support. Support is very much

related to development and maintenance, thus studying this question will answer how organisations deal with open source software development. It can also provide direct answers on how support can be improved.

What different forms of communities are there in a government environment? Do these communities differ from traditional open source communities?

In the literature study it has been put forward that communities play an essential role in open source projects. It is therefore necessary to study the communities (if any) in greater detail. Communities can be analysed using Sharma *et al*.'s framework for analysing communities. This analysis will show just how comparable government open source communities are to the traditional open source communities. Also, by studying the communities in greater detail different forms of community organisation can be encountered. Some of these forms differ from the traditional communities. Answering this question is necessary for answering the main research question: if government communities are active and comparable to traditional communities, they can play an important role in support. But even when not comparable, there can still be a role for communities.

Can communities play an important role in support of government open source projects? If so, is it needed to stimulate communities? And what is needed to stimulate these communities?

Based on the finding from the previous question, a clear answer to this can be given. If certain aspects of communities function not optimally, actions can be proposed to counter these. This is again done by looking at the analysis using Sharma *et al*.'s framework. Using the mapping the framework to Preece's definition of communities (Table 4) it can be discovered what, if any, should be improved to create real communities, or let existing communities function better. Because communities play such an essential role in support on known open source software products, this helps answering the main research question on the aspect of improved support.

Chapter 6 Case studies

This chapter provides a short description of all the case studies completed for this study. This will be a more general overview of all the projects. A deeper analysis of the functioning of all project, and their successes and failures will be provided in the next chapter. After the description of each project, the observed components of the framework are listed. These scores will be used in the quantitative analysis of the projects.

6.1 Flamingo

Flamingo is a viewer for geo-information maps. It is intended for provinces and municipalities to publish theme-based maps on their web site. It was originality developed to display risk maps for provinces, but can now display other information too.

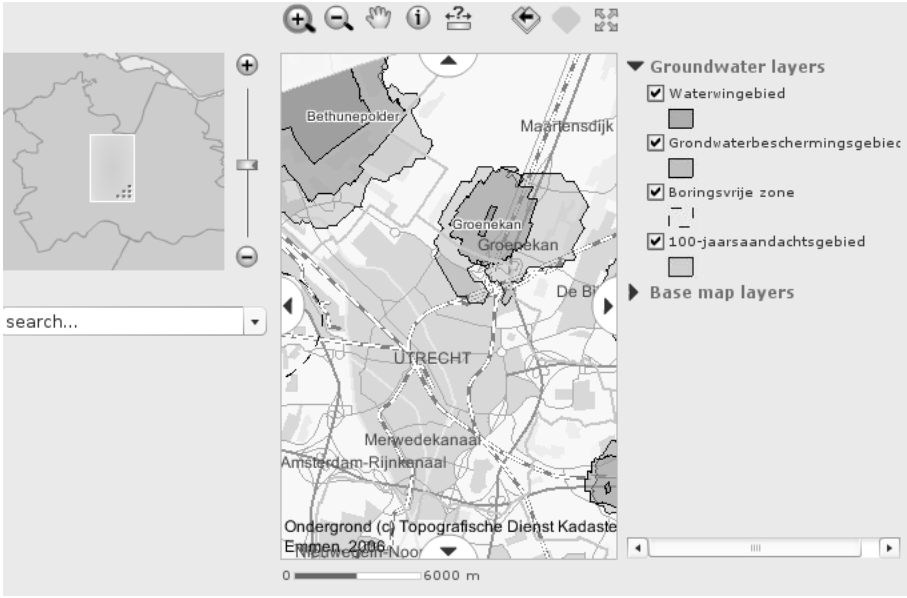


Figure 8: The Flamingo viewer in action, displaying ground water in the Utrecht region.

Flamingo is developed by one programmer from the province of Fryslân. The province had the intention to develop such an application, and had no problem releasing the application under an open source licence. This was more a pragmatic choice than a strategic one. An open source licence proved an easy way to share the application with the other provinces.

Around the project a steering committee is formed, with members of eight different provinces who use Flamingo. This committee is led by someone from the province of Groningen. This steering committee decides which features go in the application based on their needs, and coordinates development (even though development is done by only one programmer). The first version of Flamingo was very top-down managed. Although Flamingo is open source, the strategic benefits of open source were not used. Currently, version 2 is being developed. Again, this version is only developed by one programmer, overseen by the steering committee. However, this version is much more modularised. This makes it possible to add or replace components of Flamingo. The steering committee

also created a special web site to host the project. Beta versions of Flamingo are posted on this web site. Additional modules can be uploaded to the web site. This should invite other developers to make and distribute their add-ons to Flamingo.

Setting up a community web site also posted some problems. With the first version of Flamingo all users knew someone from the steering committee. When there were support questions, these could be asked easily. The new community web site is in English, in the hope to attract more international users. The web site also has a forum for support questions. However, the earlier users are still familiar with asking questions directly to the steering committee, in Dutch. Now questions must be posted to the project's forum in English. People of the steering committee should now keep an eye on the forums, and answer support questions. Even the project leader admits he is very doubtful if all committee members are motivated enough the answer such questions.

The Flamingo project has a simple community, mainly consisting of the people from the steering committee. In this committee decisions are taken in a democratic way. The steering committee holds face-to-face meetings, and the members are selected beforehand. This makes self-organisation difficult. It has negative effect on informal networks, reputation, and a flexible work structure. Because there is only one programmer doing all the work a virtual team is not needed, and reviews are not possible. The need for small additions to the code per time is also less, because coordination between different programmers is not needed. Certainly the project leader is dedicated to make the project successful as an open source project.

- Community components observed: *democratic decision making; no organisational boundaries; shared reward; motivation by altruism, ideology; electronic communication; shared trust; shared loyalty; membership (core group, community); rules & institutions; problem discovery; solution identification; release management.*

6.2 GemGids

GemGids is a geo-information system and a service for e-government in one. In a map special information can be displayed. This can range from building permits, to sewer pipes, to information about trees. This information is added as an overlay to an existing map. GemGids can work with different maps, but currently most implementations use the map service provided by Google Maps.

GemGids came forth from an initiative of the municipality of Voorst. Because more local governments were interested in the project a special foundation was formed. In this foundation local governments and commercial companies participate. Currently, GemGids has no open source licence, as defined by OSI. Only those governments and companies participating in the foundation can use GemGids. A government has to pay a fee to the foundation. For local governments this is €0.20 per inhabitant, with a minimum of €4,500 and a maximum of €9,500. After paying this fee a government has the right to use GemGids and receives support from the foundation. This includes technical support, configuration, and updates.

Figure 9: GemGids displaying building permits on a Google Map.

If a government wants something new, additional to the existing functionality, it can turn to one of the commercial partners of GemGids. For this development a government of course has to pay. Everything the company makes will be owned by the foundation and can be used by all other partners of the foundation, free of additional fees. While not strictly open source, most of the advantages of open source are preserved. The dependence on vendors is high since only those companies participating in the foundation can serve as vendor. And the results are not given back to the tax payer as free to use source code. However, it is the intention of the foundation to adopt a true open source licence once there are enough vendors, and these vendors are willing to finance all the activities of the foundation.

All local government participating in the foundation, and the commercial companies together form the community around GemGids. This community functions very much like communities seen in theory, but with a few exceptions. Because the community is closed to outsiders, members cannot freely join the community. This has its influence on how informal and flexible the community is. Because only trusted parties are in the community monitoring and sanctioning is not important.

- Community components observed: *virtual team; peer supervision; democratic decision making; no organisational boundaries; shared risk; shared reward; motivation by altruism, ideology; electronic communication; shared trust; shared loyalty; membership (core group, developers); rules & institutions; problem discovery; solution identification; code development & review; code commit & documentation; release management.*

6.3 Watlas

Watlas is the third geo-information system that was studied for this research. InterWad, through the 'Watlas', provides complete geographical data for the Wadden Sea and its

surrounding areas. This information ranges from gas pipes to locations of seals. Watlas is a special project supervised by five ministries, three provinces, and eighteen municipalities. For the Watlas the project makes extensive use of open source software, especially the Minnesota Map Server and the Typo3 content management system. Interwad is very enthusiastic about working with these open source products.

The Watlas started with a commercial geo-information system, ArcIMS. Because of licensing costs this was replaced by an open source alternative. The fact that worldwide people are actively developing the map server was viewed as positive. The positive feedback on the open source map server also opened the way for an open source content management system.

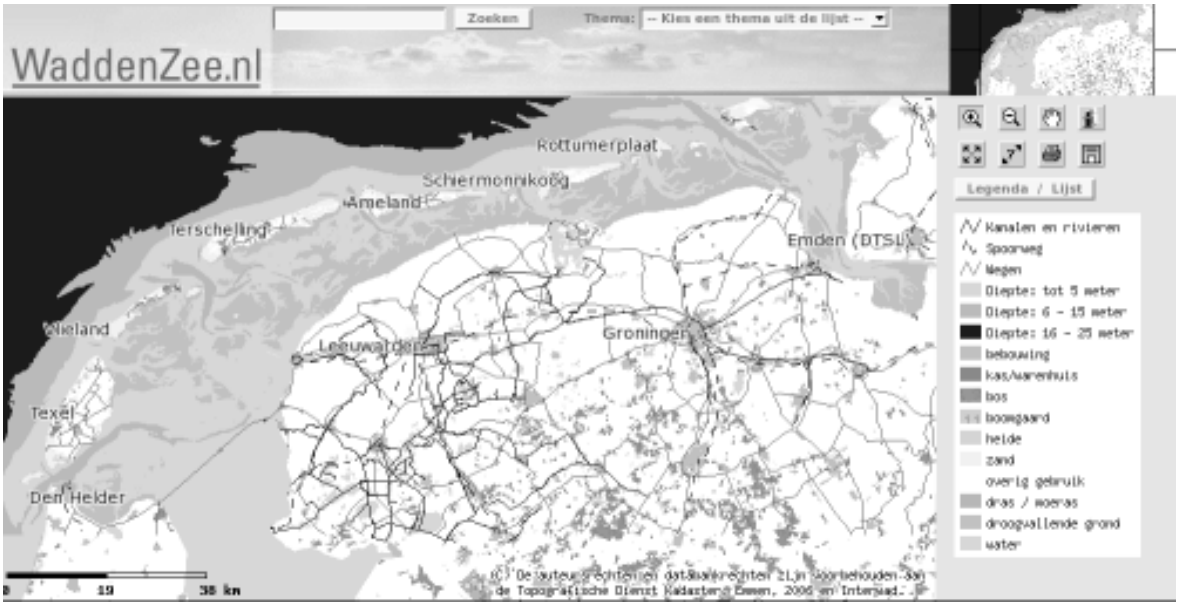


Figure 10: Watlas displaying information about the Wadden Sea

The map server and the content management system are linked to each other. This link was developed for Watlas by an external company. This link, or the whole system, is available for other government organisations. A special tool to monitor activities in the Wadden Sea area, for example of seals, is also developed for InterWad. This tool is also available under an open source licence. Besides the software, Interwad also makes all their maps and meta data available. This is done through a special community, GeoNetwork.

At the moment there is not really a community around the InterWad products. They do some promotion, but not a lot since this is not the core business. The project hopes that the GeoNetwork community will help making maps and map data better. The complete software system can also be beneficial for other organisations (using different maps), however no organisation has yet shown any interest. Even though no other organisations use InterWad's systems, there is no problem with support. Most of the software is based on existing products, and the modifications are supported by a third party. Would this party no longer be able to deliver the support, InterWad is sure other companies can take over the support of the whole system.

- Community components observed: *motivation by altruism, ideology; shared trust; shared loyalty*.

6.4 eFormulieren

eFormulieren is a central governmental programme aimed at making online forms. Many government organisations make use of online forms. This can be a simple form for asking a question or a notification of moving, to advanced forms for unemployment and social security. All forms have some things in common. Examples are checking that all data is supplied, filling in standard information, such as address based on postal code, or checking that all data corresponds with data stored elsewhere. eFormulieren builds a platform for these services and provides some standard forms to local governments.

Figure 11: An example of a form with eFormulieren

eFormulieren is based on ePlatform, an open source development platform developed by Logica CMG. The eFormulieren programme extends ePlatform and supplies the forms to government organisations. Most organisations rely on the forms made and hosted by eFormulieren. Only a small number of very large organisations make their own forms.

The initial support is done by eFormulieren. People from governments have the option to request changes or new forms. These are then made by eFormulieren. In case something is wrong with the actual program (and not just a form), eFormulieren forwards it to Logica CMG. Currently Logica CMG is the only company doing anything with ePlatform or eFormulieren. The eFormulieren programme wants other companies to be active with eFormulieren, to decrease the dependence on Logica CMG. This will be difficult because a new company will have to learn everything about ePlatform and eFormulieren before it can accept tasks.

Quality checks on the code pose another problem. Logica CMG does all the code writing, thus the code conforms to Logica CMG's standards. When other companies start producing code, Logica CMG has to check all code. For Logica CMG this whole process will also be a big change; from a competitive product to developing a product together with the competition. This is, however, something that Logica CMG is willing to do. The eFormulieren programme is asking more of Logica CMG then it can handle. Competitors can take over some of the extra business.

At the moment the community is formed by the eFormulieren programme, who acts as the project leader, Logica CMG as the main developer, and the local governments as users of eFormulieren.

- Community components observed: *virtual team; no organisational boundaries; shared reward; electronic communication; shared trust; membership (core group, developers); rules & institutions; problem discovery; solution identification; release management.*

6.5 A-select / DigiD

A-select is an authentication mechanism. It can authenticate users based on several possible means, such as a password, or biomedical data. A-select is originally developed by SURFnet, a central IT service provider for higher education and universities in the Netherlands. SURFnet administers the high bandwidth network between these institutions, but also develops several middleware applications. A-select is used by most of these institutions, public libraries, some banks, and the government.

A-select was started as a closed source product, developed by Alfa & Arris. When more and more universities started using it, there was a real need to prevent a vendor lock-in. Some universities specifically requested A-select to be released under an open source licence. SURFnet decided to buy all the code and release it under an open source licence. As a licence a BSD alike licence was chosen. Everyone could develop or use it commercially, but changes did not have to be given back to the official project. However, all big users of A-select share improvements with other users.

When the government was looking for a authentication mechanism A-select was chosen. DigiD, the service name chosen by the government, is available in two versions: one for companies, and one for citizens to authenticate. The version for companies is a bit older, and is actually a fork of A-select. This was forked even before A-select released under an open source licence. The version for citizens is newer and is the same as the official version of A-select, albeit with some small changes.

For A-select there is council of advice. This council decides on the roadmap of the project. SURFnet makes the global parts of the roadmap, but feature requests by the members are honoured. Currently everything is paid for by SURFnet. Organisations using A-select can donate features to the project, but not money. It is a long standing idea to make a formal foundation for the project, so all organisations can pay something for future developments. This foundation should be modelled on the Apache community. It is, however, still not realised.

Figure 12: Login screen of DigiD, based on the A-select software

A-select itself is maintained by SURFnet. DigiD for companies is maintained by the DigiD organisation. Because of the problems of maintaining a fork, DigiD for companies will probably also migrate to the standard version of A-select. In the past all programming work was done by the original developers, Alfa & Arris. Now more parties have experience with the code and can also program features. Such parties also give support to organisations using A-select. Support on the code is available from these commercial organisations. There is also some community support, where the developers or users answer questions. But there are not that much questions since most users have contact with a company giving the support. In case there are problems, and no organisation can fix it, SURFnet pays Alfa & Arris to solve the problem.

- Community components observed: *virtual team; democratic decision making; no organisational boundaries; informal networks; reputation as the basis of authority; shared risk; shared reward; motivation by altruism, ideology; electronic communication; global, multi cultural; shared trust; shared loyalty; membership (core group, developers); rules & institutions; reputation; problem discovery; solution identification; code development & review; code commit & documentation; release management.*

6.6 Op Afspraak

Op Afspraak (English: *On Appointment*) is an application for citizens to plan appointment with civil servants. This can be appointments for a new passport, a driving licence, or any other service provided by a municipality. By using Op Afspraak queues are reduced. Similar applications already exist, but Op Afspraak should be open source and vendor independent. Op Afspraak is developed by Atos Origen for GovUnited. GovUnited is a government programme to give municipalities a quick start with their online services.

Figure 13: Making an appointment for a passport on the web site of The Hague

An application to make appointments is the idea of The Hague municipality. This municipality is very open source minded, and wanted to share such an application with other governments. GovUnited was also interested in such a program, and a pilot group of five municipalities was formed.

Development is done by Atos Origin using the *agile software development framework*. Using this framework programs are developed in a rapid way, using fast iterations with new versions and feedback. In the case of Op Afspraak feedback on a new version is given every month. The pilot municipalities and Atos Origin also discuss new features. However, GovUnited will abandon such meetings because Atos Origin tries to avoid extra work when it notices there is disagreement in the pilot group on which feature must be implemented next. GovUnited will be a liaison between the municipalities and Atos Origin to ensure better coordination.

Once a stable version has been released the application will run on a central server. Support for the software and the server will be tendered. Atos Origin might win this tender, but that is not guaranteed. This was done to ensure maximum vendor independence. Municipalities wanting support from another vendor can get that. How coordination is done between the different parties giving support is not yet clear. GovUnited will probably maintain a central roll in coordinating the whole project.

- Community components observed: *democratic decision making; fluid political coalitions; shared reward; motivation by altruism, ideology; electronic communication; global, multi cultural; shared loyalty; membership (core group, developers); rules & institutions; reputation; problem discovery; solution identification; code development & review; release management.*

6.7 Modernisering GBA

Modernisering GBA (Gemeentelijke Basisadministratie; English: *Municipal population register*) is a central government project for a new version of the GBA. This version will replace the current commercial versions offered by different vendors. Modernisering GBA (mGBA) will develop the central database for all data, and the services to read and write data from and to the database. The interface is not developed by Modernisering GBA, but will be left to commercial vendors.

The software developed by mGBA will be open, but it will not be open source software. Everyone can look at the code, but no changes to the code are allowed. This is done to simplify support. Everything that the GBA must do is directly related to the law. Since the law is the same for everyone, no municipality should need extra or adapted functionality. While not being open source itself, mGBA uses open source software as much as possible. This is done so all the software can be easily implemented at the municipalities, without requiring each municipality to pay high licensing costs for the software. Around mGBA there is not a community, but mGBA participates in the communities of the software products they use. They submit bugs and/or fix some bugs themselves when they find one.

Once mGBA is finished it will run on a central server. Support for the whole application is given to a special governmental organisation. Municipalities will not deal directly with mGBA, but use an interface developed by a vendor. It is left up to the vendor whether or not the interface is also open, or if the software is even open source software.

- Community components observed: *peer supervision; shared reward; motivation by altruism, ideology; electronic communication; shared trust; shared loyalty; membership (core group, developers); rules & institutions; problem discovery; solution identification; code development & review; code commit & documentation; release management.*

6.8 MMProject

MMProject is a document system based on the open source project EGEMProject. Both products are built on the framework of the open source content management system MMBase. EGEMProject was chosen as a starting point because it had the desired functionalities. Improvements were made by a commercial vendor, resulting in MMProject.

MMProject is now very stable. It runs on the servers of the internet service provider, who also gives limited support. The original vendor is no longer needed because all the bugs have been solved. A second release of MMProject was planned but never conducted. The new release would have incorporated feedback from the users. However, all users were satisfied, making a second release unneeded.

eOverheid is the only organisation running MMProject. They wanted to give the code back to the community, and contacted OSOSS for extra information on how to do that. This has never resulted in an actual release to the MMBase community. There were not enough contacts between the community and eOverheid. And not enough information about the licence that should be used was available. eOverheid does not want to be liable, and wanted to release the software “as-is”, so there were many questions about the most appropriate licence. During development other organisations had contacted eOverheid for more information, but none has ever requested the code.

Another problem with MMProject is documentation. This is very minimal. Within eOverheid only a few persons know enough about MMProject. Only the original developer has technical knowledge about the project, but never documented that. This can also hinder successful adoption by other organisations, or continued use within eOverheid.

- Community components observed: *none.*

6.9 APLAWS+

APLAWS+ is a CMS (Content Management System) initiated by London Borough of Camden. APLAWS+ was initiated in 2001 as part of the LAWs project. This was a project by the Office of the Deputy Prime Minister to create local authority web sites (LAWs). Local authorities received funding from the national government to create better web sites. Camden choose to adapt a current CMS, Red Hat CMS. Right from the beginning Red Hat was a partner in the adoption of their CMS, to make it better suitable for local authorities. First versions were adapted versions of Red Hat's CMS, where changes from the one were merged into the other, and vice versa. Because this was not a practical situation it was decided to separate the projects and develop each on its own (thus making a fork). Camden had not enough programmers to develop such a large project completely alone. Through informal contacts between APLAWS+ project members and people from West Sussex County, this county joined the development of APLAWS+. West Sussex put twenty programmers on the job. Not all programming work however was done by parties who use the product. Through tendering several features were added by commercial companies. Some tenders were even won by a company from India, proving that making an application available under an open source licence can create interest worldwide. All these companies were required to release their work under the same open source licence.

- Community components observed: *virtual team; peers supervision; democratic decision making; no organisation boundaries; informal networks; fluid political coalitions; reputation as the basis of authority; shared risk; shared reward; motivation by altruism, ideology; flexible work structure; electronic communication; global, multi cultural; shared trust; shared loyalty; membership (core group, developers); rules & institutions; monitoring & sanctioning; reputation; problem discovery; finding volunteers; solution identification; code development & review; code commit & documentation; release management.*

Chapter 7 Analysis of the results

In the previous chapters all case studies were briefly described, and the scores of the communities on the framework of Sharma *et al*. were listed. This chapter will provide an overall analysis of the projects based on the theories from the earlier chapters. The analysis will not be done per project. It is not the intention of this report to judge individual projects, but rather to see on which qualities all projects perform well, and which are less dealt with. Part of this analysis will be done by quantifying the results from the interviews (see also Appendices C and D), the other part is a 'softer' analysis, based on common themes in each interview. In the end conclusions about support for open source projects will be drawn.

The first part of this chapter looks at the more general observations about the studied projects. After that, the projects can be analysed using the framework by Sharma *et al.* (2002). This gives information about how projects themselves are organised. It is also important to look at the communities in the projects. It will become clear that communities can perform different roles, and that there are different forms of communities. These different forms will be important for showing the different roles a community can take in support of the software. To even better understand communities, the last part of this chapter looks at communities in greater detail. Again using Sharma *et al*.'s framework communities can be analysed, to show what aspects (if any) are missing in current communities. For this last analysis the quantitative results are used.

7.1 Overall practices of the projects

7.1.1 Support

All projects have currently made arrangements for the support. There is some variety in how each project deals with support. In all but one project, support is handled by a commercial company or companies. The fact that the software has an open source licence was no issue for all the different companies providing commercial support. Only in the case of Flamingo there is no company giving support, instead support is dealt with by the users and developers of the software. Overall, no support issues in the studied projects were found.

7.1.2 Communities

A second observation is that all interviewees responded that communities are important. There are currently not many fully established communities. However, there is the recognition that communities play an important role in the development and use of open source software. Ultimately, all projects want a community. The interviewees are however, uncertain on how to form a community. The result of that is that some project have some sort of a community, or started forming a community, while other project have nothing of a community.

7.1.3 Open source culture

A third observation is that all interviewees are enthusiastic about open source software. There is a drive to succeed as an *open source* project, not just as a *software* project. This is an important observation. It shows that these people have something that can be called an open source culture. This is needed to succeed as an open source project. A point of concern is that this enthusiasm for open source software is mainly present with the project

leaders, not within the whole community (or potential members of the community). Many interviewees responded that some organisations are very sceptical about open source software. For them open source software still has an image of "long-haired hacker software" that cannot be taken seriously.

7.1.4 Licences

Another important observation is that not all projects that were studied have an open source licence. GemGids and Modernisering GBA do not have an open source licence, according to the Open Source Definition by OSI (OSI, 2007). For these projects this was an intentional choice. Yet, these two projects were studied, and provide valuable insights. While not strictly open source in licence, these projects look like open source projects. These projects have most, or all of the benefits of open source software. Not using an open source licence yielded other, additional benefits. GemGids is only "open source" for participants of the foundation. Others cannot view, use, or adapt the code. Participants of the foundation freely share everything. This leaves the benefits intact. By limiting access to participants, and asking a fee for being a participant, extra support can be offered. In literature this is called a *gated community* (Perens, 2005), because it is only a true community once a gate (in this case being a participant of the foundation) has been passed. Modernisering GBA is also not open source software. While everyone can view the code, none may adapt it. This was done so there is no proliferation of the software, and support is a lot easier.

Other projects said they are open source in licence but do not actually publish their sources, namely MMProject and Watlas. Watlas will only distribute their source code to other government organisations. With such a restriction Watlas cannot be called open source, since is fails the definition for open source licences, specifically "no discrimination against fields of endeavour". MMproject is in no way required by any open source licence to actually publish the sources. While not publishing the source code, the project can still have an open source licence, and thus technically be open source. However, not publishing the source code takes away many of the advantages of open source software. With such restrictions development is hard to follow for outsiders, which can hinder the adoption of the software.

7.2 Project organisation

The organisational framework by Sharma *et al*. (2002) for analysing communities can be used to analyse how the project as a whole is organised. Although not all projects have active communities, the four dimensions of the framework (structure, culture, governance, and development) are very well suited for the analysis of the organisations running and/or using the software, and the intentions they have with the community. Because the framework for analysing communities is based on a framework for analysing organisations, this is very easy. After the analysis of the projects, the complete framework can be used again to analyse the communities. In this first analysis of the projects, it is also sometimes useful to make use of the more detailed components of the framework, especially when a dimension is not dealt with in a best manner. It is then useful to look at the more detailed components in each dimension.

7.2.1 Structure

The structure defines how an open source project is made up. This includes the people and organisations that participate in the project, how participants are involved, and how all involved parties make decisions.

Number of organisations using the software	Number of vendors supporting the software	**One**	**Multiple**
One		2	0
Multiple		4	3

Table 5: Number of case studies where the software is used by one or more organisations, and development is done by one or more organisations.

To analyse the different participants in the project, it is important to make a distinction between organisations using the software, and organisations developing the software. Table 5 shows these results.

In a majority of the cases studied, the software is used by multiple organisations. However, most of the software projects are only developed and supported by one organisation. The fact that most of the software is shared between organisations is a good sign. Some projects are not picked up by other organisations. It might be that their promotion is not good, or that there is simple no interest in the software, for example because there are alternatives in the market. That most projects are being developed by one organisation does not have to be a problem *per se*. It can however be a problem when the project grows. All of the projects that are now being developed by multiple organisations started as being developed by just one. This created problems in finding other organisations willing to code additional features. It takes time for new organisations to acquire the necessary knowledge. Another problem that is identified from the cases studied is quality checks on the code. The original vendor often sees the code as theirs, and wants to make sure the program remains stable and competitive. To make sure of that, the original developer wants to examine the code from other developers before actually committing code to the software. New commercial companies therefore have higher prices and need longer times to complete an assignment than the company who knows the code throughout.

Decision making is dealt with in a pragmatic way. In most projects this is quite democratic. There are meetings where important decisions are taken. All projects with a community organise face-to-face meetings for important decision making. In such meetings everyone can request new features, or discussions are held on what features to include in the next release. Projects also discuss many things via electronic means. For example, within eFormulieren users can e-mail feature requests. When this seems to be a sensible request, or when there is high demand for it the decision is taken to implement the feature. However, the project leader(s) have the ultimate decision. Even when many users request a feature, it might still not be implemented. This is not much different from a project as Linux, where one person decides which new features go in the kernel, and which are rejected (Ingo, 2005).

7.2.2 Culture

The culture of an open source project is very important. All participants should have a clear understanding why the project is open source and what the exact purpose of the project is. Other important aspects of culture are how people interact with each other, and how they communicate.

In all case studies the interviewees had a good understanding why their product is open source. By releasing it as open source software they thought a number of benefits were reached. Two reasons were heard most. The first is that by releasing it under an open

source licence the code can be easily shared with other organisations. Releasing it as open source is a quick and easy way to do this, without the need for complicated technology contracts between organisations. A second reason is licence costs. This can come in two forms: reduced licence cost for the own organisation, or not forcing other organisation to pay huge licence fees. Many interviewees responded that using open source software was cheaper for their organisation. The interviewees also responded that they used other open source software because of this reason. This implies that the total cost of ownership is indeed lower for open source software than for closed source software. For the studies projects the debate on that issue (Knubben, 2004) favoured open source software. The other argument dealing with licence costs comes when distributing the software to other organisations. Some projects have the purpose of making software for other government organisations. These projects do not want to force the receiving organisation to pay high licence fees for the extra software needed to run the open source software. All projects also have good understanding of what their software should do and how their software differentiates from other, similar software. The other benefits of open source software as named in Chapter 4 were hardly mentioned.

The studied projects are all local projects, in the meaning of 'not global'. Projects exist for a specific reason, and up front the developers think about the intended users. This intended users are often reached, but organisations outside these bounds are difficult to reach. This is also very difficult to achieve. When a project is first intended for the Dutch market, the software and all documentation will be in Dutch. To reach more global users the software and documentation must be (partly) available in English. For many projects this is a huge step, one that also brings with it many risks: the software might not be picked up by international users, resulting in a wasted effort for translating everything. Only in Flamingo and A-select there was a drive to go global. All other projects wanted to keep the project local.

Flamingo struggled with this issue. For version 2 a special web site was created, with a forum for support questions. All users should use this forum. This means that all long-time members (the provinces) must now use English to communicate with each other. Members found this very unnatural, when all users can also read Dutch. Many community members kept sending e-mail messages in Dutch to the project leader, asking for support. The result is that the questions and answers are not archived or visible for others.

A-select choose from the beginning to do everything in English. Because this was done from the very start, users are used to communicating in English. While A-select is in English, international use of A-select is not very high. Only in the Netherlands A-select is the dominant software for authentication. An important reasons for this is DigiD: when the government chose to select DigiD, many other organisations followed. Knowing that even the government uses it gives certainty.

7.2.3 Governance

Governance is related to the rules a project has on letting people in, banning people, and formal structures around a project. All projects use more or less formal structures for their decision making. This can be a council of advice or a formalised user group. These groups meet at regular intervals to discuss the future direction of the project. These are always face-to-face meetings. The exact role of these formal groups differs per project. In some projects this group makes final decisions at a very detailed level. In other projects the group just sets global directives, and leaves precise implementation and smaller improvements to individual members writing the code, or consensus between the developers.

Membership of communities is not very diverse. This has, for example, consequences for *monitoring & sanctioning* and *reputation*, two components part of governance. In most projects these two components are absent. This is because there is not a diverse group of developers monitoring each other, or a large user base that has active discussions that needs to be monitored. When there is not such a diverse group, reputation plays no significant role. Only in larger communities, members who are really active can distinct themselves from other members and earn a higher reputation.

7.2.4 Development

Development relates both to the development of the software, and to that of the community itself. In the majority of the cases studied development is good. However, there are some issues with projects where multiple parties develop the code, such as A-select/DigiD, and in the near future eFormulieren. New code needs to be added to the central code repository. The first developer must allow others to add code, trusting that other parties can produce high quality code. Interviewees from these projects responded that the developing party always wants to check code made by other developers, before adding the code to the actual application. This review of other parties' code takes time. Progress is slower, because one organisation has to do all reviews. When not all new code has to be checked for quality by just one organisation, development can be quicker. For this, developing parties must trust each other, even if they are competitors. One developer from the community, other that the original writer of the code, should do the quality-control on the code (regardless of the organisation he works for). This way organisations sometimes check code of their own employees, sometimes of employees from other organisations. Nevertheless, it ensures that code is checked for quality by another person than the original writer.

The process of finding new users is very different between the projects. Half of the projects do not recruit actively, while the other half do just that. One half release the software and hope to attract new users by having a good name in the market. Exposure of the project, on internet sites, magazines, conferences and more certainly helps. The other half have employees to find new users for the software. This is done for projects that are started by a central government organisation and are meant for lower governments. A prime example of this is eFormulieren. The lower governments for which the software is intended need to be convinced of using the software. New organisations can join the project. They way new organisations join (asked or joined themselves) determines how the project functions.

7.3 Communities in government open source projects

Communities already play a role in the organisation of each project as was described above. But since communities play such an important role in open source projects in general that is it worthwhile to look at communities in greater detail. From the literature it is clear what roles a community can play in an open source project. Not all the roles as detailed in the literature can be found that explicitly in the current communities. The roles the current existing communities have is somewhat different. Also the form a community can take is different from the forms as encountered in the literature.

7.3.1 Current roles of a community

During the case studies, not many fully functioning communities were found. An explanation for this is that most of the projects are fairly new. Building a community takes time. But signs of communities taking shape are certainly present in most projects. All projects ultimately want a full community.

The communities that exist now perform different roles. What roles the community

currently provides depends on who is a member of the community. In a majority of the projects the community is a collection of user organisations. At regular intervals people from each organisation hold face-to-face meetings to discuss current affairs of the project. In some communities the developer is present at these meetings, but some communities prefer to first reach a conclusion internally, and then instruct the developer. Because all projects hold face-to-face meetings, these meetings are also used for knowledge exchange. Only in the case of the Flamingo community, user-to-user support can be observed. A reason for this is that the Flamingo project is the only project without a commercial support partner.

Communities are also used for pooling resources. Different organisations participate in one project, and each organisations can donate something to the project. This can either be money or code. By donating money to the project a developer can be hired. Or the organisation can have an own commercial support partner. The organisation can then instruct that company to build something, and donate the result to the community, so all can use it.

When there are multiple users and one developer, it can be called a community. In that sense a majority of the projects have a community. These communities are however different from the typical communities known from the literature. How each interviewee thinks his community should look like differs for each project. This is the right attitude for a project that wants to form a community. There is no single mode of organisation that can be applied to all projects. Projects need to grow in ways that suit the particularities of the project. This also includes how the community should function. A number of projects reported that they want two communities: a developer community, and a user community. They fail to see how this can be one single community. By truly integrating the developer community with the user community, users can 'move' to a different level within the community. This corresponds with the onion model presented in Chapter 3. Only with such an integrated model all of the community benefits can be enjoyed. Overall, communities are not yet used for development and maintenance by many different parties.

7.3.2 Different forms of communities

Because there is the understanding that communities can play an important role in support, all projects aim to extend and strengthen the current community, or create a community. Not all projects aim for the same kind of community. From the case studies three different forms of communities can be identified.

The first from is the *gated community*. Projects that have this form are GemGids, and Modernisering GBA. GemGids is truly a gated community. Modernisering GBA only protects the source code from modifications. While everyone can take a look at the code not everyone can join the community. Other parties are also prohibited to take the source code and make another product out of it. Like GemGids, this is done to guarantee better support. The fact that these projects are not open source does not degrade the project. This form of community fits these kinds of projects best. Also, a gated community can evolve over time in an open source community, when restrictions to join the project are dropped.

The two other forms of communities are real open communities. But they differ in how they are formed and structured. The first can be called a *formal community*. This is a community around an open source project where the project leader (or team) works hard to create that community. The project actively searches for new organisations to use the software, and thus be part of the community. Interactions in the community are formalised. Meetings are planned at regular intervals. There is an institution that governs the whole project. Such a form of a community can be observed at eFormulieren, Op Afspraak, and

A-select/DigiD.

This is a clear distinction with what can be called the *informal community*. Like a formal community this is a community around an open source project, but these communities rely more on self organisation. This does not mean that there are no formal meetings or institutions, but the way they act is less strict or forced. APLWAWS+ and Flamingo are the only projects that have such an informal community.

The type of community can have consequences for what type of support can be delivered. It is however, important to note that one form of community is not better than another. All forms have their advantages and disadvantages. It is important that a project has a community that suits the project, and that changes can be made if necessary.

7.4 A quantitative perspective

The analysis in the previous paragraphs was made by looking at common themes from the interviews, and linking that to the theories on open source projects Sharma *et al*.'s framework (2002) divides the four dimension (structure, culture, governance, and development) in a great number of components (see Figure 4 on page 18). Projects can be ranked on these components, so a quantitative analysis can be conducted. With this quantitative analysis is is possible to show what aspects of a community are underdeveloped, and should therefore receive more attention.

The components are treated as dichotomous variables. A variable is either true or false. This results in the dataset shown in Appendix C. Care must be taken when analysing these results. Since only nine cases were studied it is too few to draw statistically significant conclusions from this quantitative approach.

Table 8, in Appendix D on page 80, shows the percentages of projects where each variable was observed. There is a great range in these variables. Some are only encountered in very few projects, while others are present in almost every project. When these numbers are combined with the framework some interesting point emerge (see Figure 14 in Appendix D on page 81). In this figure the bold items are those variables that have the highest scores, while the italic items are those with the lowest scores. It is noteworthy that successful and unsuccessful variables are mixed between the different dimensions. Only the dimension 'structure' contains no variables that belong to those with the highest scores.

An even clearer picture emerges when these results are overlaid using Preece's definition (See Table 9 in Appendix D on page 82). This can be done using Table 4 that was used to link Preece's definition to Sharma *et al*.'s framework. Variables with the lowest score link to 'people' and to 'policy'. Variables with the highest score link mostly to 'purpose', 'computer systems', and again to 'policy'. The aspect 'people' is greatly underdeveloped, while 'purpose' and 'computer systems' are sufficiently developed. 'Policy' gives a very mixed picture, suggesting that here too things can be improved as well.

The groups that are underdeveloped are 'structure', 'people', and 'policy'. Any actions to improve the workings of communities should therefore focus in improving those three things. However, not everything can be steered. If the variables from 'structure' with the lowest scores are observed, something interesting is visible. These variables relate to an informal, fluid, and flexible network. These are indicators that cannot be directly steered. These properties of a community only emerge over time, mostly by self organisation. Looking at variables that are the most successful also gives an interesting conclusion. Here the variables can be split into two groups. First there is the group of things that can be

organised directly. This is how the community functions, the rules in the community, who is a member, and how releases are dealt with. The second group are variables that are almost a precondition to any open source project: motivation, trust in other members, and loyalty to the project. These are also very difficult to steer, it is therefore encouraging that these indicators are very strong.

When looking at all the variables with a low score in the concept 'people', it can be seen that these all relate to how the community is formed. It means that the people working with communities are not able to get these values in the community they try to create. Yet, all interviewees are enthusiastic about open source software, and want to form communities. But for well-functioning communities, more people are needed. This result shows that other people in the community – and therefore the community as a whole – find it difficult to deal with such concepts as informal and fluid networks. Whether these difficulties are with other government organisations, or from commercial companies part of the community is not known. However, members of the community are uncertain about the community in general, or lack enough knowledge and experience with open source software and communities.

Another interesting conclusion that can be drawn when looking at 'policy'. Some of the variables in this group have the highest score, while others have a very low score. The ones with a low score (peer supervision, monitoring & sanctioning, reputation) or with an intermediary score (code development, review, code commit & documentation) can be resolved. Most of the projects do not make any use of specialised software for development, such as version control systems, bug trackers, etc. Making better use of such systems could improve this, and ultimately strengthen the community as a whole.

The dimensions in the framework are influenced by all other factors. This can be seen by looking at the arrows between the three major parts in the framework. This means that changes in one influence changes in the others. For example, when the structure of a project changes (more informal, for instance), this will affect the culture (more flexible work structures, or more electronic communication, for example). It will also affect the process: governance must change, and the way development is conducted will likely change too. For the management of a project this means that improvements can be made in any part of the framework. As a consequence all other factors will change too. This is an important concept since some components are easier changed (more manageable) than others. It also means that not all issues have to be dealt with directly. Some of the more manageable ones can be changed. These changes will have their effect on the other components as well.

In conclusion, it can be said that the results from this quantitative approach support the results from the earlier analysis. With all issues identified, recommendations can be given to improve support, improve the projects, and improve the communities.

7.5 Conclusions

All the case studies were analysed by looking at common features in all projects and by comparing the projects to projects known from the literature. All projects can get support in ways that are most appropriate for the project. For some projects this means support in cooperation with a commercial partner, while for others this means community support. While support has currently been good, projects expect future support issues when the project grows. There is a strong believe that communities can play an important role in supporting the project. However, there is great uncertainty about how to create such

communities.

With the organisations studied there is not yet a broad 'open source culture'. While the project leaders are enthusiastic about the project as an open source project, many other do not know what benefits there are for the project as an open source project, and what role they could play in the community.

Related to that is that not all project have an open source licence. While for some projects this is done intentionally to provide better support, for others it is uncertainty about the benefits of open source licences. For example, some organisations only share code with other government organisations, while the potential benefits could be greater when such restrictions are dropped.

When it comes to development and maintenance, most projects are still developed in a classical way. Only one commercial company does all the development, and this company handles all support. No other companies develop on the software, or provide support for government organisations.

The communities around government open source projects are a bit different from traditional open source communities. One such community is described in literature: the gated community. Within this community, only members can view and use the software. There are requirements for being a member, but once a member everything is shared. Such communities are formed to guarantee support.

There are two other forms of communities. Both are true open source communities, but how they are created differs. The formal community has a central organisation that plans the whole community and actively seeks new community members. For the informal community, this central organisation is not present, and more is left to self organisation by the members.

Chapter 8 Next generation support

All projects have been analysed in the previous paragraph. This shows the state of support and communities at the moment. But to propose recommendations to improve support, a vision about the use of communities in support must first be established. This chapter presents such a view. First, the different roles a community can perform in support are described, linking roles to the current characteristics of government communities. After that the focus is shifted to improve the current projects so the visions in the first paragraph of this chapter can be met. The chapter concludes with some recommendations on how new projects, or projects without a community, can build a community.

8.1 Support from communities

The previous chapter analysed the current state of open source projects, and the communities that exist now. While there are currently no support issues, problems are expected when projects grow. All organisations think communities will play a big role in future support. The question is what role this can be exactly, and what the impact is of the different forms of communities found in this study.

What support is can be seen by looking at the ASL framework presented in Chapter 1. Most of the actions that need to be done in the ASL framework must be performed by the organisation themselves, and communities cannot play a role in it. Paragraph 3.4 presented a modification of the ASL framework, one in which a community is present (See Figure 7 on page 24). Only in development and maintenance a community can play a role. Besides support, a community can also give user-to-user assistance. Because only these two actions can be performed by a community, these are the only ones that are studied in greater detail.

8.1.1 Development and maintenance

What role a community can perform in development and maintenance depends on how the whole project is organised, and on the characteristics of the software.

Different situations of using open source software

When development is done by an external company, that company has a strong connection with the software. Any community will most likely include the original developer. That company then becomes an active member of the community. This means that the developer keeps influence on the software and development thereof. This still means there is a strong dependence on the original developer. To reduce this dependence, new companies must join the community. During the case studies interviewees reported that it is often difficult to find other companies willing to join the community. The original company has to authorise new companies direct access to the source code. The original developer has to agree other companies are co-developing, which undercuts the position of the original developers. This is seen as a potential problem in some projects. Development and maintenance by a community can only be truly successful when multiple organisations and commercial companies participate.

If the software is developed or adapted in house, community development is more likely. A government organisation would have less hesitation to allow co-developers to work on the software. This can be seen in the case of Flamingo. Although at first development was done by just one developer, by setting up a project page where others can contribute code, the project shows it is opening up development. It is not a primary task of the government

	Community development	Commercial development
Use of the software	Usable for large, mature, generic open source projects. In large enough communities organisations can make use of user-to-user support.	Usable for all sorts of open source software. No support problems are to be expected as long as the commercial company has the necessary knowledge.
Source the development	Difficult to manage, because the project must be very successful before the sourcing organisation can close formal ties with the developing company and rely only on the community.	The developer stays involved with the product. It is difficult to find new companies, beyond the original developer, willing to support the software and become part of the community.
Develop internally	Usable if the product becomes successful. Then, new companies are willing to join the community. If not, then one or a few persons remain responsible for all support questions, which is a big risk to all involved organisations.	Used when governments adapt an open source project. The development to adapt to specific demands is done internally. But because the software is based on an existing product, a commercial party has knowledge about the original product, and can do the support.

Table 6: The type of support is influenced by how an organisation uses open source software

to develop software, so any co-developers reduce the time spent on development per organisation. This leaves more time for the government organisations to focus on their primary tasks. This is a real incentive for organisations to source development and maintenance to a community when development was done in house.

When the software is not written from scratch, but instead is an adaptation of existing open source software done in house, the support can very well be given to an external company. This can be seen in the case of APLAWS+. This project is based on the existing Red Hat content management system. Any company already familiar with this system can also support APLAWS+ because the adapted version still shares much resemblance with the original product. When the supporting company already knows the original product, it will be able to support the adapted product.

Table 6 list the different forms of using and developing open source software, and when community development or commercial development is possible.

The role of different forms of communities

If a project has some sort of a community, this community can be used (to some extent) for development and maintenance. The extent to which this can be delivered depends on the type of community. In projects with an informal or a gated community this co-development will be best. Here, users have the strongest association with the software, and are willing to invest time in it.

Projects that have a gated community created such a community with the intention of guaranteeing support. Users can join the gated community, pay some fee for this, and get the software and support on that software in return. In projects where there is a formal

	Advantages and disadvantages
Gated community	*Advantages*: ● Because only members have access to the software, it is always clear who uses the software. ● By asking a fee for membership, support can be paid for for all members. *Disadvantages*: ● The development is closed for outsiders, therefore the community will not grow very rapidly. This results in fewer potential developers.
Formal community	*Advantages*: ● There is a clear leader for the community. ● The leader has an interest in the continuation of the community. This gives security for all members. *Disadvantages*: ● Members will look at the leader first when something needs to be done. Members are less likely to take up tasks.
Informal community	*Advantages*: ● Members organise their own community and decide for themselves what is best for the community. *Disadvantages*: ● Much depends on the motivation of the members. If only a few members are motivated this could be a risk for the community. If one of these members quits, there can be continuation problems.

Table 7: Types of communities and their advantages and disadvantages

community, an interesting problem arises. Community members will first look to community initiators when additional development or maintenance is needed. Community members see this person or organisation as the leader of the community, and members show some dependence on this leader. Also, within a formal community, there is less incentive for the individual members to really get to know the software, and become an active member of the community. The community organiser performs this role, and because it is organised, there is a guarantee that the community organiser will keep this role for a certain amount of time. Because it is an official task to organise a community, incentives for others to do something are lower.

In projects with an informal community, things are least sure. These projects have leaders too, but these emerge because they started the project, or have a high reputation. Within an informal community, there is always the understanding that everyone should do their task to keep the community going. But because of this uncertainty, there will be incentives for others to invest time in the code, and help others solve problems. An informal community has its risks too. If no one becomes active in the community, development will remain with just one party, leading to a risk for the community as a whole. If this person decides (or must) stop development there are no others to take over the development.

Table 7 summarises the advantages and disadvantages of the different types of communities

With such support issues, the question is what advantages has community support for open source software over commercial support on closed source software? While it is indeed true that there are potential issues and risks with open source software support, in the studied projects the benefits outweighed the risks. In closed source applications, only the original developer has access to the source code, creating a vendor lock-in for the organisations using the software. Using community support every organisation and every commercial company can join the community and offer support on the software. The support through a community is what sets open source software apart from closed source software. By allowing others to join the community, software can develop faster, and there can be competition on support, leading to lower prices. These mechanisms are not possible with closed source software. Support on open source software from a commercial company is a mix between community support and closed source support. Whether or not this has benefits depends on the community around the project. The benefits will be greatest when there are multiple companies in the community, all able to give support on the software.

8.1.2 User-to-user assistance

Lakhani and Von Hippel (2001) detailed how users of open source software give "free" user-to-user assistance. For many (non-professional) users this form of support can be more than enough. For many large communities the speed at which questions are answered is very high, and many problems are solved by community members. Thus, besides development and maintenance, a community can also play an important role in user support.

In the studied projects, such user support is hardly encountered. Only the Flamingo project has no commercial support partner and relies only on the support community members give to each other. In practice this means that the community leader and developer answer most of the questions. A reason for this large lack of community user-to-user assistance is that not all layers of the onion model (Figure 3 on page 13) are full of users. Most communities consist of project leaders, developers, and some active users. However, none of the studied projects have a very large user base yet. Therefore there are not many users requesting assistance. Having larger communities, with community members in more diverse roles, creates potential for user-to-user assistance. Having user-to-user assistance in a community is a big advantage for the project because problems with the software are easily identified.

The reason that the community members do not form a diverse group is twofold. First of all, most of the projects produce software that is not directly targeted at end users, but rather at system administrators. These system administrators are part of the development layer of the onion model. Secondly, when software is targeted at end users, these end users are not much involved in the community. Many people using open source software do not even know that the software is open source and what they could contribute to that. By informing users that they use open source software – and why they use it – this involvement can be improved. However, care must be taken when all potential users of the software are invited to ask for user-to-user assistance. As users will have questions, there should also be users who can answer those questions. They should know enough about the software to actually do so. To achieve this the community should have a sustainable grow path. Growing too fast creates only members in the outer layers, and no members with enough knowledge in the inner layers. However, when to growth is too slow, the project can stall, or not take off at all.

8.2 Towards better open source projects

During the analysis in the previous chapter, and the vision in the previous paragraph, room for improvement in projects was identified. A total of five important points were found:

- uncertainty about open source software;
- no broad open source culture;
- not everything is open;
- only one supporting party;
- no co-development.

Many of these points deal with open source software, not just support on open source software. However, if the total level of open source practice increases, the level of support a community can give will increase.

8.2.1 Uncertainty about open source software

A first problem is uncertainty about open source software. Although the general ideas of open source are known, some more advanced topics are not well understood. This is especially the case for knowledge about communities and community building. Other knowledge that is often missing is which open source applications other organisations are using. The knowledge that is missing can be split in two: explicit knowledge and tacit knowledge. Because of their characteristics both types of knowledge require other means of transfer.

Explicit knowledge is knowledge that is well understood and codified. It can be easily transferred electronically. The reader can easily assimilate the knowledge by reading it. This explicit knowledge is general knowledge on what open source software is, why one should use it, who uses what, and the advantages and disadvantages of open source software. This knowledge can easily be managed centrally. A central government organisation (such as OSOSS for the Dutch government) should be responsible for transferring this knowledge to other persons working for various government organisations.

By definition tacit knowledge is much harder to transfer than explicit knowledge. The tacit knowledge consists of knowledge on how to manage open source software, and how to create and manage open source communities. Tacit knowledge needs to be transferred using personal training or communication. However, this knowledge can also be partly transferred using electronic communication. As Roberts (2000) writes: “individuals cannot share tacit knowledge effectively [by electronic means] unless they share a common social and cultural context. If this condition is fulfilled they may *share* tacit knowledge by assimilating codified knowledge and thereby creating new tacit knowledge that will be largely, though not completely, the same.” (p. 435) This means that people can share experiences on how to deal with open source software, or on how to create communities, over a mailing list or an online forum. Here, tacit knowledge is written down in an e-mail message or forum message, turning it into explicit knowledge. Although this is not all available knowledge, it is sufficient for the receiving party to reconstruct the tacit knowledge. Thus, there is a synergy between the explicit knowledge in the message, and knowledge the receiver already has. One condition for this to work is that sender and receiver must be able to understand each other. As Roberts note, this can only be the case when they both have a common cultural context. That is, when they both have already some knowledge on open source software and communities.

To sum it up, information is very important and should be transferred by both central organisations and decentralised: persons dealing directly with open source software, support, and communities. However, for this to work, there should be a common open source culture.

8.2.2 No broad open source culture

A culture for open source software and development is important. People are more enthusiastic when they know and approve of the concepts of open source projects. Commercial companies are far more willing to support and release their software under an open source licence when they have faith in the mechanisms involved. And users of different organisations can better understand each other if they know what they are talking about.

Users need to get familiar with open source software. Large open source projects can play an important part in this. Most users do not choose, say Firefox, Apache, or MySQL, because it has an open source licence. They choose it because it is good software. But while using it they get familiar with open source software. It is therefore important to stimulate the use of mature open source projects. This stimulation should not only be targeted at decision makers – that is people who decide what software is used in an organisation – but also at end users. End users should be aware that they are using open source software and that they could participate in a community. Of course, not every end user would do that, but some might, and these opportunities should be taken. In conclusion, it is not only necessary to stimulate the use of open source software, but also use of the mechanisms behind open source software.

8.2.3 Not everything is open

Not all projects encountered during this study are open source software. The licence model under which they distribute their software is not compatible with the Open Source Definition from OSI (OSI, 2007). When done on purpose, and with reason, this is a very valid position, and should not have to change. Other projects that were encountered are open source in licence, but do not promote this very well, or do not open their development. This undermines many advantages of open source software, and could be an extra hurdle for new adopters. These projects have not much to lose, but much to gain when opening their source code. When a project is determined to be an open source project, it should do that fully. That means publishing the source code, show development (intermediary code), have an open forum or mailing list, etc. This should be as easy as possible. The current government platform “Uitwisselplatform” (comparable to SourceForge.net) is a good start. However, the current website leaves some things to be desired. It would also be good if development on the Uitwisselplatform was better promoted. For example, the use of Uitwisselplatform could be one of the requirements in a tender.

8.2.4 Only one supporting party & no co-development

Currently, most of the software projects studied rely on a single party for their support and additional development. While it might be very convenient dealing with only one party, lock-in effects are likely. In an ideal situation the community around an open source project should consist of multiple companies plus all organisations using it. To reach this goal, companies have to be invited to join the community. New programming tasks can be given to new companies willing to join the community. This way, new companies gain experience with the code base. Selecting a new company for the first time will be more expensive, because the company would have to familiarise itself with the code before programming can begin. However, for future programming tasks, there are multiple

companies to choose from, where competition will lead to lower prices. Because most programming tasks are additional tasks, the prices will not be above the threshold for European public tenders. Government organisations can invite any company for new jobs without violating European law.

Another option would be for new companies to voluntarily join a community. In many large open source projects this can be seen. Also in the case of AWPLAWS+, the most mature project studied, this has happened. But this is something that cannot be forced by policy. Commercial companies will join an open source community when such a community is attractive enough. This means the software should be widely used and already have an active user community. There should also be broad government support for open source, giving enough insurance so companies are willing to invest even without formal contracts. However, in the current climate, this scenario is still far off.

8.3 Creating communities

Communities are important for any open source project. Although an open source project without any form of active community is possible, it takes away some of the attractive advantages over closed sourced software. Sharing information can be beneficial for all involved parties, and a community is an excellent way to share information (and development). The relevant question is how to create such communities? And exactly who should be part of the community?

This research has shown that most communities within the government are different than the typical open source communities as described in literature, or encountered in large open source projects. Within most communities there is a single company doing all the work, and multiple government organisation profiting from this. All these organisations are giving input to the developers. Often, government organisations do not deal directly with the developer. Instead, one government organisation has a coordinating role. When one organisation is using open source software, and has contact with a commercial company, they could also function as the first leader of the future community. It should therefore be the responsibility of this organisation – along with organisations already using the software – to find new users who will install the software and join the community. Especially for low-profile projects this is the most easy way. New organisations can be found by informal contacts. There are links between different government organisations, and if tasks are somewhat equal, information on how to do these tasks can be easily shared. For projects that are high profile, it can be beneficial for the commercial company to find new users. Finding new users ensures that the company can do additional implementations and get support contracts.

Another important question is whether there should be a role for organisations such as OSOSS in fostering new communities. In an ideal situation: no. Government organisations should realise that sharing is beneficial, that creating communities helps other organisations, but also themselves. Nevertheless, central organisations helping other organisations set up communities can be beneficial in the beginning. Much of the knowledge about setting up and running a community is tacit knowledge. And if information about communities is not present within many organisations, a central organisation is needed to spread this tacit knowledge. Once the level of knowledge among all organisations reaches a certain level, community building should be left to the involved organisations. Would a central government organisation always help, the resulting communities would be formal communities, where all involved parties keep looking at the

central organisation for help. Delegating tasks would then be a problem.

There is some research on building communities (e.g. Fogel, 2006; Goldman and Gabriel, 2005). Although government communities are different from open source communities outside government organisations, there is no reason why these general theories about community building would not work. However, a definitive to-do list for setting up a community cannot be given. Even if it could, this would not be beneficial. Every community requires its own growth plan, and needs to form according to its needs. Setting up every community to a pre-designed plan would most likely result in a failure for most communities.

The frameworks used in this report do give some insight it what must be present when building a successful community. Trust between members is essential. And willingness to succeed as an open source project. If there is no will to succeed as an open source project, or if (future) members show no interest in community dynamics, the community will fail. In an open source community trust is very important. Tasks must be delegated and members need to trust each other that tasks are done in a reasonable fashion. Because every community relies on voluntary action, there are no mechanisms to force other members to do certain tasks. Without trust between member, tasks cannot be delegated.

When setting up a community, the first question is, who should take the lead? Should this be the first government organisation who wants a particular program, or the commercial company who writes and manages the code? For both organisations a community can be beneficial. It therefore can be a task for both parties. Both parties will have different contacts that might be interested in the program. Both parties should search for other organisations that might have an interest in the program. Using special sites such as the Uitwisselplatform, the project can be advertised.

Unless there are specific reasons to create a gated community, the community should be as open as possible. There is no harm in showing the source code, or showing the communications of the project members. This allows everyone to view the workings of the community. By showing an active community, others will be inclined to join the community. Once organisations are willing to participate in the community, expectations are important. These should not be too high, but also not too low. A community cannot, and will not take care of all support questions. Having a community does not guarantee that all support is taken care of, or that development will continue. However, too low expectations can result in doing nothing for the community. Overall it is important to trust the community member's instincts when it comes to building and running a community. Past research (e.g. Van Wendel de Joode (2005)) and this research again has shown that communities are self organising. Even though people are uncertain about how to build and run a community, when they do build one the end result is a good functioning community.

Chapter 9 Conclusions and recommendations

This final chapter will present the conclusions of the report, answering the main research question, and giving recommendations on how government policy towards open source software support and communities can best be organised. Finally, as any report, this report will have its shortcomings, and these will be discussed, along with recommendations for further research.

9.1 Research findings

9.1.1 Main research question

In the introduction, and again in Chapter 5, the main research question was stated. The research question is:

In what different roles are government organisations involved in open source software development, how do they deal with open source software development, and how can support for this software best be organised?

There are different roles a government organisation can have in open source software development. These roles can be divided into two categories: active roles and passive roles.

Passive roles have those organisations who only participate in a community. This can be a large, existing open source community outside the government, or a government community. Government organisations can make use of well-known open source software. When reporting bugs or posting feature requests they become part of the community. Another passive role is being a member of a government open source project. Here the role is as a member, often in meetings about the developments of the project.

Organisations can also have active roles. An organisation has an active role if it is developing for the community, or trying to build one. The organisation can be the leader of the community. This is especially the case for those organisations trying to set up a formal community. Another active role is sourcing the development to a commercial party. Whether or not the organisation keeps this active role differs per project. The organisation can keep this leadership role, or it can become passive, leaving the role of community leader to the commercial organisation.

Regardless of the role an organisation has in a community, it should think about support for the software. How support must be organised does not depend on the role in the community. Most tasks that must be done as part of application management have no relation with the community and should be done by the organisation using the software. Every organisation running the software should do the tasks as defined by the ASL framework. These tasks are not dependent on the fact that the software has an open source licence. For software with a closed source licence, many of the tasks set forth by the ASL framework are the same. A community can only play a major role in two aspects. First, regarding information. A community, or any sort of contact between organisations, helps spreading of relevant information about open source products. Second, a community can play an important role in development and maintenance. This is the only part of the ASL framework where a community has any influence. When multiple organisations are part of a good functioning community, the development and maintenance of the software can be

shared by all organisations. In larger communities government organisations can also profit from user-to-user assistance. Here, users help each other by answering questions. However, whether or not this functions well depends on a large enough user base, and parties willing to spent time on answering questions from other parties.

Development of the software is a part where a community can play a major role. The fact that the software has an open source licence sets it apart from closed source software development. Other than closed source development, the organisation can have influence on future development. This can be a major plus for an organisation. To have any influence an organisation should be part of the community. Influence comes with authority, and authority can only be gained by doing something for the community. The development itself can be done in two ways: either the software can be developed by a government organisation, or it can be left to a commercial partner. Either way, a community is important and should be set up. This could be done by the government organisation, or by the commercial partner, but ideally by the two parties together. Together they form the first seed of the community.

Participating in a community is beneficial for all parties using the software. A community will not form if there is either no interest from other organisations in the software, or when the software is badly promoted. When no critical mass is reached, the project can not be successful as a community supported project. It could be that the software is very specific to a task not done by any other organisation, or there can be already other (open source) software for the task. During this research one project was encountered where there is nothing of a community. There was no interest from other organisations, and no real interest by the organisation running the software to search for other organisations willing to use the software.

The fact that a community can play a major role in support of the software is the biggest advantage over closed source software. Because the community can also consist of multiple commercial parties, an organisation is not bound to a single commercial vendor. For commercial support, the organisation can switch to another vendor that is part of the community, or even invite other commercial parties to join the community. When the community is very successful the organisation even has the choice to entrust its support wholly to the community. By opening software to multiple government organisations and commercial parties, an organisations get a far greater freedom in how support is dealt with.

The conclusions from this report are in line with earlier research on governments and open source software (e.g. Schmitz et al., 2004), but adds a new dimension to it. While earlier research mainly focussed on the idea that governments can profit by using the well-known, existing open source software, this research shows that it is also possible for government organisations to actively participate in open source development, even when projects are small in size. It also shows that such participation helps to organise support on the software.

9.1.2 Case finding

None of the studies projects reported large problems in supporting the software because it was open source software. All project could get support in ways they want to. Yet, many of the interviewees think that open source software is fundamentally different from closed source software. This leads to uncertainty about open source software, although it does not limit how the software is used.

A real open source culture is missing in most of the studied organisations. Although project leaders are enthusiastic about open source software, people working with the

software often do not know why they use open source software. This limits the creation of a fulling functioning, heterogeneous community.

Other things that were noted during this research is that some projects only share code with other government organisations. Or that the code has an open source licence, but is not published. Making the software available to everyone can be beneficial for the organisations. Only when the code is broadly available can all advantages of open source be enjoyed.

While many projects are open source, the development is mostly done by a single company. There is no real co-development between different organisations. In addition, for most software only one commercial company provides support on a commercial basis.

Although there are not many open source communities, the communities that do exist show different characteristics than traditional open source communities. When looking at open source communities in a government setting, three types of communities can be found: formal, informal, and gated communities. The gated community produces no *open source* software because the software is only accessible for members who are accepted into the community. In a formal community there is one organisation leading the community, actively promoting and stimulating the community. Informal communities rely more on self organisation. All types of communities have their advantages and disadvantages. When creating a new community, the type of community must be chosen. Switching of community type is possible. The structure and workings of a community can change over time. This can be intentional, or an outcome of self organisation when other community members get different roles. However, the organisational structure of an open source project is not the only thing that influences the workings of a community, many other variables are of influence. Many of these variables are explained by Sharma *et al.*'s framework for communities. A government organisation should try to implement as many to have a good functioning community.

9.2 Recommendations for government policy

If the central government wants all government organisations to use open source software, it should stimulate its use. To have the best support for all these new open source programs, communities should be created. However, communities function best when there is as little external control as possible; community members should build their own community. Therefore the government should only create an environment where open source software has the potential to function fully, where government organisations are free to choose the software they want and can create communities with vendors and other government organisations as they see fit.

For support in general, not that much has to be done. Support of all studied projects has so far been good. For new organisations adopting open source software, it is important to know that support issues on open source software are the same as for closed source software. Open source software is not “a league of its own” when it comes to support. Open source can fit into the existing support operations within many government organisations.

One of the most important things that should improve is information about open source software and communities. There is uncertainty about open source software. Sharing information between government organisation – such that organisation can learn from each other – can help solve this problem. A central mailing list especially for government

agencies could help solve this problem. By sharing information about software use, open source products, or experiences with communities, fear or uncertainty can be reduced. Information from a central authority is another good solution, but should be temporary. Information sharing and participating in communities is a task of government organisations themselves, not a central organisation. One task especially suited for a central government organisation is to facilitate contact between organisations. There should be an informal network between government organisations where information can be shared. Setting up such a network can be a good task that can be conducted by a central organisation.

Another problem is that many employees of government organisations do not know what open source software is, or that they – or their organisation – work(s) with open source software. This information should be shared with employees. Employees should also be stimulated to join the community. People who work with the software on a daily basis know best how the software can be improved. This way, end users in government organisations can be fully part of the community.

A future problem with support can be that many open source project have only one vendor who does the support on the software. This can lead to future support issues, when vendor lock-in can occur. Open source software can only prevent this vendor lock-in if all aspects of open source software are fully used. The source code should be open to everyone. There are existing tools to facilitate this. Uitwisselplatform (based on GForge) is such a tool, but use of it is minimal. Most of the projects studied here do not use this, or any other platform to publish their source code. First, there should be more stimulation to use such a platform. It forces a project to be as open as possible. For example, use of Uitwisselplatform can be a requirement when sourcing a task. Secondly, the Uitwisselplatform itself can be improved. Some functions do not work (such as daily code snapshots, browsing source code, anonymous code checkout), and others can be improved.

Although no real solution can be offered for language issues, this is something every project has to think about actively. When a project is created it must decide what the target audience is. This has large influences on the potential users, and on the critical mass. If it is decided that all materials are in Dutch, only Dutch organisations will be interested. Organisations not fluent in Dutch cannot find or understand the software. On the other hand, if it is decided that all materials are in English, it can alienate other Dutch organisations because they prefer Dutch. Doing everything in multiple languages would seem a nice solution, but takes effort, without much guarantee beforehand.

9.3 Discussion

This report analysed how government open source projects handle support, and how this support can be improved. Now that this question is answered, some points of discussion can be raised.

First, there is a limitation in the number of projects that were studied. Nine projects is not an awful lot, especially when quantifying those results. However, these were all the projects that could be found, that also met the condition that the project needed to code their own software, not just use existing open source software. Not all projects are open source in licence, yet even those projects provided valuable insights.

Secondly, the way in which both Preece's definition for online communities and Sharma *et al.*'s framework were used to analyse the results also have a number of limitations. This is the first time these are used to study and analyse open source projects. It is not known what

the results would be if the same definition and framework are to be applied to familiar open source projects. When results from the analysing using this method is different from other analyses of these projects, the methods here are not very well suited for the analysis. However, it is likely that such projects will pass on all the components in the framework and adhere to the definition by Preece.

In Sharma *et al*.'s framework projects are only ranked in a binary way. Each variable in the framework is either true of false. This makes clear a distinction between practices in some projects difficult. Also, important components are missing from the framework to analyse *software projects*. The framework assumes a project is open source, making analysis of projects that are not very open, or do not have an open source licence difficult. The framework presumes code is available and tries to analyse how the surrounding community functions. When there is no such a community analysis is difficult. A third aspect that is missing from the framework is knowledge. Knowledge about how to start, organise, or run an open source project is important. In the current framework there is no variable that directly relates to the amount of knowledge people, or the organisation as a whole, posses(ses). Yet, this research shows that knowledge is lacking, or at least that there is great uncertainty about own open source projects. This can also be related to the fact that the framework presumes an project is an open source project, meaning that the basic knowledge should already be available.

Another limiting factor in this framework is that all components are treated as equal. The analysis presumes the importance of every variable is the same. Clearly, this is not the case. Some components are more important than others. For example, “membership” and “finding volunteers” is important, without which an open source project cannot be successful. On the other hand, a project can be very limited in scope (“global, multi cultural”) yet successful. For a good analysis this relative importance should be known. For this research the relative importance was not used on purpose. It is not clearly established which variables are most important and which are not. Furthermore, it is likely that the importance of variables is different between 'government open source' and 'regular open source'. Asking ten experts for their opinion will result in ten different answers.

A third point of discussion is how the data was gathered. Interviews were held with key persons for each project. While they can provide valuable insights in the project, they view the project in a more favourable light, since it is their project. Interviewing more users and community member might give other insights.

9.4 Recommendations for further research

Many studies have been done on open source projects. However, always successful projects are studied. This is inherent in what they study, because only projects that are alive can be studied. While some historical analysis can be made on those projects, the reasons why these projects succeeded and other not cannot always be found. The projects studied for this research are very diverse. Some projects are large, and can be considered successful. Other projects are just beginning, and their success or failure is yet unknown. It would be very interesting to study these same projects in a year's time, or in three years. Some will become successful, while others fail. Using the information collected in the report, on the first steps of these projects, and their ideas on how to create communities can be very valuable for the further study of open source projects.

This report mainly focussed on support on open source communities in a government setting. One of the main conclusions is that communities can play an important role in this.

For this it is necessary to create successful communities. While there is literature on creating communities, and this report has some information on how to create communities, more information is needed on exactly what actions are needed to create such communities.

Appendix A Components in the open source framework

The framework for analysing open source communities lists a total of 25 components (Figure 4 on page 18). The explanation for each component is listed here.

Multi site virtual teams
Projects members are scattered over different geographical locations. It team working on the project is not a physical team, by a virtual team, working from many different locations, only communication by electronic means, not using face-to-face meetings.

Peer supervision
Different users work on the same code. The guarantee quality code should be checked and tested. Not only by the persons who writes the code, but also by the other people working on the project. This checking by peers ensures that there are less errors in the code, and keeps programmers motivated to write good quality code, because their peers will look at the code.

Democratic decision making
Communities rely on self-organisation. People can join the community and people can leave. Community member's roles can also change when they become more involved in the project. This can only be successful when people have the feeling that their contributions are taken seriously, that their input matters, and that they can have an influence on the direction the project takes. For communities it is important to listen to anyone, and not have a “dictator” who controls the whole project and does not listen to other members. If this would happen, and community members have the feeling that their contributions are not taken into account they will start their own project.

No organisational boundaries
An open source project should not be confined to just one organisation. If only one organisation would use the software there is no real merit in trying to form a community. The whole software can be easily supported by the own organisation, without going through all the trouble of having a community.

Informal networks
Self-organisation implies informality. When a community if very formal, everything is define beforehand, self-organisation will not work.

Fluid political coalitions
A community should not consist of different camp, with their own opinions on where the project should go. Instead, there should be a general idea of the project's purpose. Of course there will be discussions on what to implement next, of what solution is used to solve a problem. However, community members should decide the position based on each case individually. In effect, every time a decision needs to be made, community members decide their stand, forming a coalition. This coalition favours their preferred solution, while other community members favour theirs. When a decision is reach these coalitions dissolve again. If two groups of the same members are always against the other's solution, it might be better to split the project.

Reputation as the basis of authority
Decision making should be democratic, but that does not mean that everyone's opinion should be treated as equal. Long standing community member have better knowledge of the project, and there should have a better understanding of what is the preferred solution.

When new members become very active, these members get a higher reputation, and this reputation should be translated into a higher authority. However, this authority should not be formalised, instead community members should know instinctively which members are influential.

Shared risk and ownership
No community member really owns an open source project. If different members are working together on the same project, these members all own part of the project. At the same time, both members share the same risk by using the project. In a community it is important that things are shared equally, i.e. not one member should take all the risk while others have no risk.

Shared reward based on merit
All project members should share the rewards of the project. One community members should not profit more from the work of the whole community than any other.

Motivation by altruism, ideology
To succeed as an open source project, it is important to have the drive to succeed as an open source project. Setting up an open source project can be difficult, certainly in the beginning. Keeping everything closed and confined to one organisation is easier. However the reward of being open source comes much later. A project can only succeed of the first members are truly motivation in being members of an open source project.

Flexible work structure
The structure of the project should be flexible, so that self-organisation can work best.

Electronic communication
Electronic communication is important for every open source project, certainly when project members are scattered over different locations. When electronic communication (e-mails, forum messages) are saved and available online, this also serves as a good introduction for new members. People can see how the project function, and get some feeling for the culture of the project. This can help decide whether or they they want to join the community.

Global, multi cultural
In an ideal community, the software should appeal to many people.

Shared trust
Community members should trust each other. Without trust cooperation can not happen.

Shared loyalty
Just as members should be motivated to succeed as an open source project, all members should have some loyalty to the project. Without shared loyalty, there is no guarantee that members will keep working on a project, and this is not a good basis for trust.

Membership (core group, community)
The onion model (Figure 3 on page 13) shows that there are different roles of community members. All these roles should be present. Of the project only consists of developers and no users, the project will not come very far.

Rules & institutions
Within any organisation rules are needed. These rules govern what behaviour is considered appropriate for community members, who can submit code, or when a new version is released.

Monitoring & sanctions
Rules need to be monitored, and when rules are broken, sanction can be applied. In a good functioning community this will not happen very often, but it is important that serious rule breaking is dealt with accordingly.

Reputation
There should be community members with a reputation that distinguishes them from others. Such a reputation is important for governance, because someone must be able to make decision quickly when needed, without being questioned to much afterwards by the rest of the community.

Problem discovery
Progress should be made (partly) by problem discovery. All community members can report a problem with the software (a bug, or a missing feature), and this should trigger an action from the developers.

Finding volunteers
A community where all the members are static is not a good community. It is natural that member's roles change. The community should attract new interested people to the community, to guarantee its own survival.

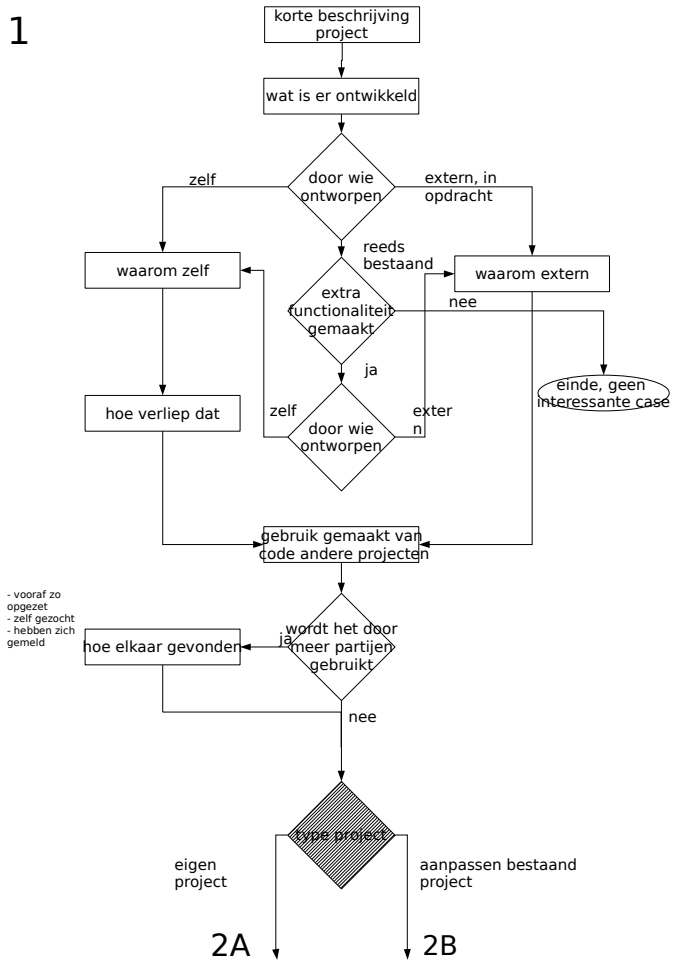
Solution identification
When a solution is brought forward by non core developers, the developers should be able to recognise the solution, review it and implement the solution in the code. Developers should not write all the code themselves when there already is a solution from someone else. This can also be a solution for a similar problem in another open source project. Overall, it is important to not duplicate efforts.

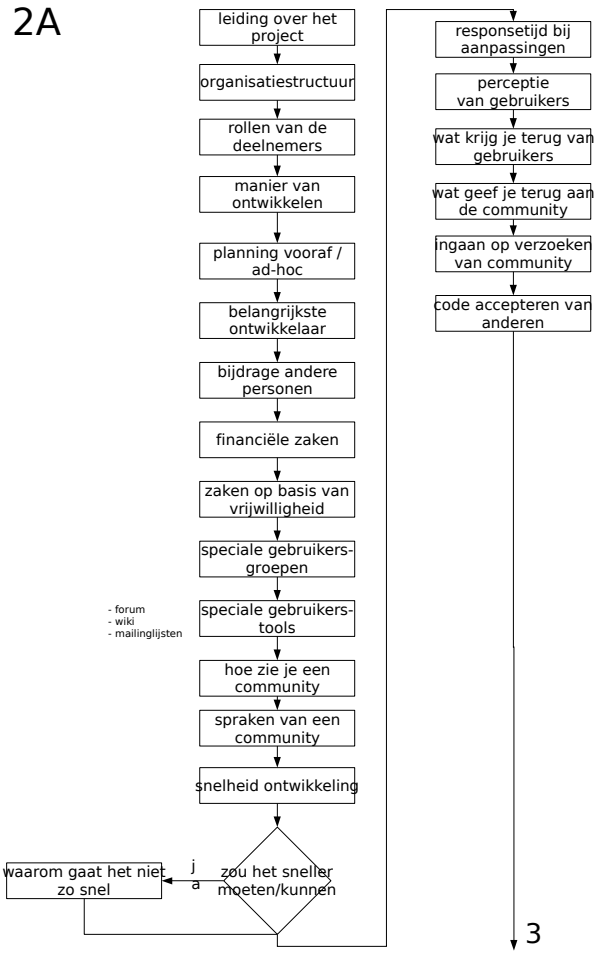
Code development, review
When code is developed, it should be reviewed by many people. For this to work effectively, code should be added in small quantities. When very big changes come at once, no one can see all its complication.

Code commit & documentation
When code is added, there should be documentation of that code, so that everyone reading the code can understand what the code does. Even if much later something in that code needs to be changed, the documentation will tell exactly what that piece of code does. Documentation helps to better understand and manage the code.

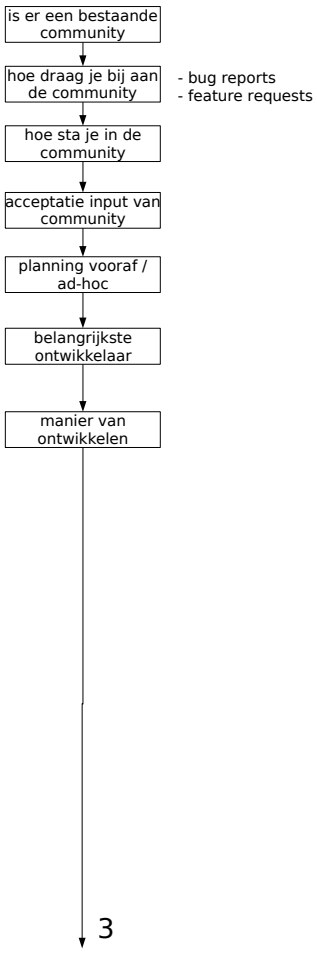
Release management
An open source project should have policies for release management. It should be clear when a new version will be released, who prepares the release, when features are considered good enough for a release, what version number the release will have, and how the release will be published to all users.

Appendix B Interview protocol

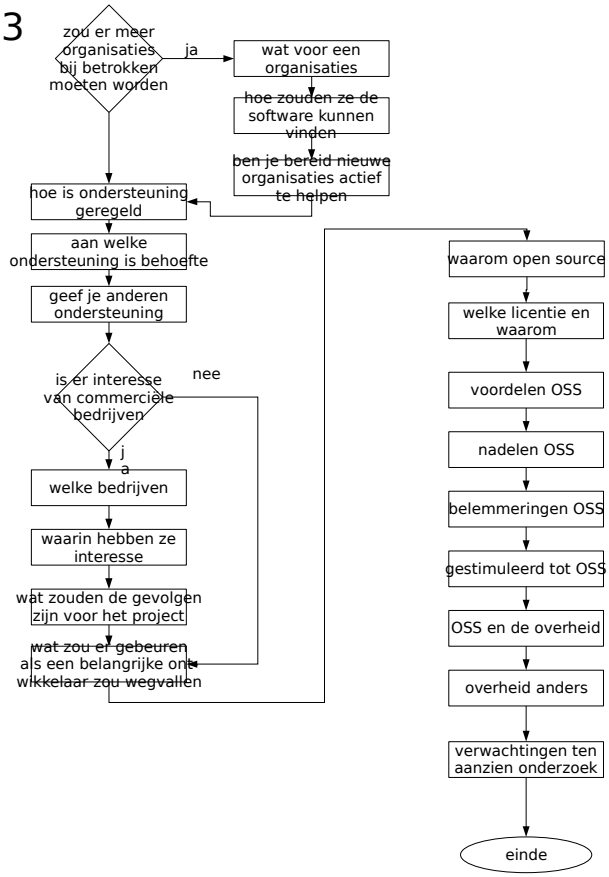



2A
leiding over het project
organisatiestructuur
rollen van de deelnemers
manier van ontwikkelen
planning vooraf / ad-hoc
belangrijkste ontwikkelaar
bijdrage andere personen
financiële zaken
zaken op basis van vrijwilligheid
speciale gebruikers-groepen
speciale gebruikers-tools
- forum
- wiki
- mailinglijsten
hoe zie je een community
spraken van een community
snelheid ontwikkeling
zou het sneller moeten/kunnen
j
a
waarom gaat het niet zo snel
responsetijd bij aanpassingen
perceptie van gebruikers
wat krijg je terug van gebruikers
wat geef je terug aan de community
ingaan op verzoeken van community
code accepteren van anderen
3

2B


is er een bestaande community
hoe draag je bij aan de community
- bug reports
- feature requests
hoe sta je in de community
acceptatie input van community
planning vooraf / ad-hoc
belangrijkste ontwikkelaar
manier van ontwikkelen
3

3


zou er meer organisaties bij betrokken moeten worden
ja
wat voor een organisaties
hoe zouden ze de software kunnen vinden
ben je bereid nieuwe organisaties actief te helpen
hoe is ondersteuning geregeld
aan welke ondersteuning is behoefte
geef je anderen ondersteuning
is er interesse van commerciele bedrijven
nee
ja
welke bedrijven
waarin hebben ze interesse
wat zouden de gevolgen zijn voor het project
wat zou er gebeuren als een belangrijke ontwikkelaar zou wegvallen
waarom open source
welke licentie en waarom
voordelen OSS
nadelen OSS
belemmeringen OSS
gestimuleerd tot OSS
OSS en de overheid
overheid anders
verwachtingen ten aanzien onderzoek
einde

Appendix C Dataset

virtual_team	,00	1,00	,00	1,00	1,00	,00	1,00	,00	,00
peer_supervision	,00	1,00	,00	,00	,00	1,00	1,00	,00	,00
democratic_decision_making1	1,00	1,00	,00	,00	1,00	,00	1,00	,00	1,00
no_organisation_boundaries	1,00	1,00	,00	1,00	1,00	,00	1,00	,00	,00
informal_networks	,00	,00	,00	,00	1,00	,00	1,00	,00	,00
fluid_political_coalitions	,00	,00	,00	,00	,00	,00	1,00	,00	1,00
reputation_bases_on_authority	,00	,00	,00	,00	1,00	,00	1,00	,00	,00
shared_risk	,00	1,00	,00	,00	1,00	,00	1,00	,00	,00
shared_reward	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	1,00
motivation_by_altruism	1,00	1,00	1,00	,00	1,00	1,00	1,00	,00	1,00
democratic_decision_making2	1,00	1,00	,00	,00	1,00	,00	1,00	,00	1,00
flexible_work_structure	,00	,00	,00	,00	,00	,00	1,00	,00	,00
electronic_communication	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	,00
global_multi_cultural	,00	,00	,00	,00	1,00	,00	1,00	,00	,00
shared_trust	1,00	1,00	1,00	1,00	1,00	1,00	1,00	,00	,00
shared_loyalty	1,00	1,00	1,00	,00	1,00	1,00	1,00	,00	1,00
membership	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	1,00
rules_and_institutions	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	1,00
monitoring_and_sanctioning	,00	,00	,00	,00	,00	,00	1,00	,00	,00
reputation	,00	,00	,00	,00	1,00	,00	1,00	,00	,00
problem_discovery	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	1,00
finding_volunteers	,00	,00	,00	,00	,00	,00	1,00	,00	,00
solution_identification	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	,00
code_development_and_review	,00	1,00	,00	,00	1,00	1,00	1,00	,00	1,00
code_commit_and_documentation	,00	1,00	,00	,00	1,00	1,00	1,00	,00	,00
release_management	1,00	1,00	,00	1,00	1,00	1,00	1,00	,00	1,00

Appendix D Analysis of the results

Variable name	Percentage observed
Multi-site virtual teams	44%
Peer supervision	33%
Democratic decision making	56%
No organisation boundaries	56%
Informal networks	22%
Fluid political coalitions	22%
Reputation as the basis of authority	33%
Shared risk & ownership	78%
Shared reward based on merit	78%
Motivated by altruism, ideology	50%
Flexible work structure	11%
Electronic communication	67%
Global, multi cultural	22%
Shared trust	78%
Shared loyalty	78%
Membership (core group, community)	78%
Rules & institutions	78%
Monitoring & sanctioning	11%
Reputation	22%
Problem discovery	78%
Finding volunteers	33%
Solution identification	67%
Code development, review	56%
Code commit & documentation	44%
Release management	78%
	n = 9

Table 8: Components from the open source framework and the percentage of projects where that variable was present

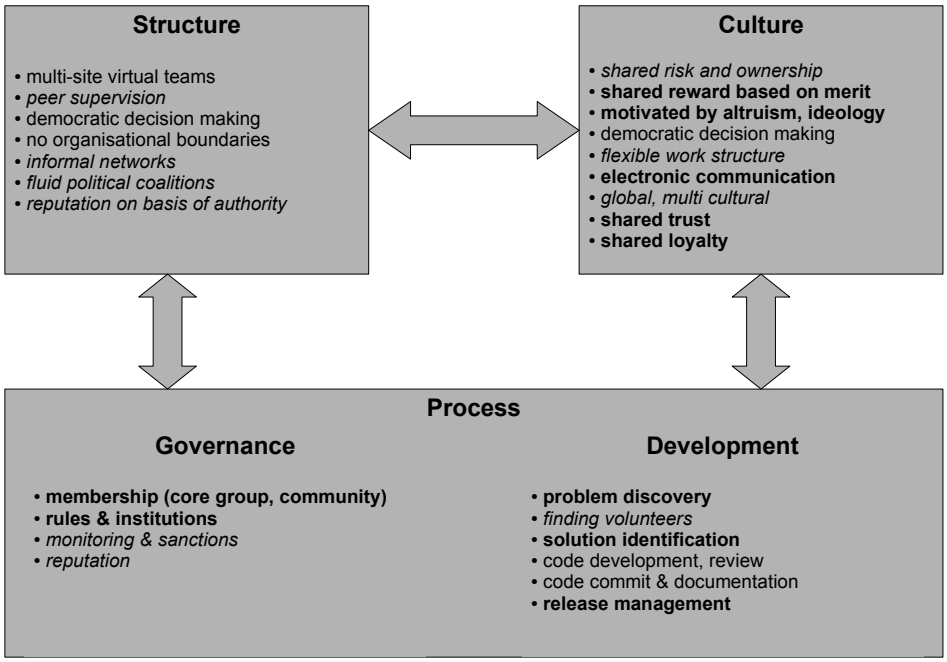


Figure 14: The open source model with variable scores overlaid

People	**Policy**
Multi site virtual teams	*Peer supervision*
No organisational boundaries	Democratic decision making
Informal networks	*Shared risk and ownership*
Fluid political coalitions	**Shared reward based on merit**
Flexible work structure	**Rules & institutions**
Global, multi cultural	*Monitoring & sanctioning*
Membership	*Reputation*
Finding volunteers	**Problem discovery**
	Solution identification
	Code development, review
	Code commit & documentation
	Release management
Purpose	**Computer systems**
Shared risk an ownership	**Electronic communication**
Shared reward based on merit	Code development, review
Motivation by altruism, ideology	Code commit & documentation
Shared trust	**Release management**
Shared loyalty	
Problem discovery	
Solution identification	
Code development, review	
Code commit & documentation	

Bold: belonging to the 1/3 of the highest variable score
Italic: belonging to the 1/3 of the lowest variable score

Table 9: Preece's definition with variable scores overlaid

Appendix E Bibliography

Applewhite, A. (2003). *Should governments go open source?*. IEEE Software, 20 (4), 88 – 91.

Baarda, D.B., De Goede, M.P.M. and Teunissen, J. (2005). *Basisboek kwalitatief onderzoek: Handleiding voor het opzetten en uitvoeren van kwalitatief onderzoek*. Groningen: Stenfert Kroese.

Baarsma, B. *et al*. (2004). *Kosten en baten van open standaarden en open source software in de Nederlandse publieke sector*. Amsterdam: SEO.

Bonaccorsi, A and Rossi, C. (2003). *Comparing motivations of individual programmers and firms to take part in the open source movement. From community to business*. Retrieved on 15 January 2007 from http://opensource.mit.edu/papers/bnaccorsirossimotivationlong.pdf

Britzer, J., Schrettl, W., and Schröder, P.J.H. (2004). *Intrinsic motivation in open source software development*. Retrieved on 3 September 2007 from http://opensource.mit.edu/papers/bitzerschrettlschroder.pdf

CCW Research (2005). *Linux, Windows NT, and Windows 2000/2003. TCO research in enterprise computing*. Retrieved on 10 April 2007 from http://download.microsoft.com/download/9/2/C/92C833BA-9F23-4580-8678-740812556CF2/CCW_TCO_report.pdf

Cybersource (2004). *Linux vs. Windows. Total cost of ownership comparison*. Retrieved on 10 April 2007 from http://www.cybersource.com.au/about/linux_vs_windows_tco_comparison.pdf

Derksen, B. and Noordam, P. (2006). *Modellen die werken. Kwaliteit in bedrijf en informatievoorziening*. Gouda: Boekdrukkunst uitgevers.

Evans, D.S. and Reddy, B.J. (2003). *Government preferences for promoting open-source software: a solution in search of a problem*. Michigan Telecommunications and Law Review, 9 (313).

Fogel, K. (2006). *Producing open source software*. Sebastopol, CA: O'Reilly.

Forbes (2005). *The open source heretic*. Retrieved on 12 April 2007 from http://www.forbes.com/execpicks/2005/05/26/cz_dl_0526linux.html

Hahn, R.W. (Ed.) (2002). *Government preferences for open source*. Washington, D.C.: AEI-Brookings Joint Center for Regulatory Studies.

IDABC (2007). *OSS (open source software)*. Retrieved on 1 May 2007 from http://ec.europa.eu/idabc/en/document/2627/5644

Ingo, H. (2005). *Open life: The philosophy of open source*. Retrieved on 20 August 2007 from http://openlife.cc/online

Kay, D.G. (1992). *A course in computer law*. ACM SIGCSE Bulletin, 24 (1), 252 - 254.

Knubben, B.J. (2004). *Investeren in openheid. Een analyse van TCO-onderzoeken betreffende open source software*. Den Haag: OSOSS.

Lakhani, K.R. and Von Hippel, E. (2001). *How open source software works: "free" user-to-user assistance*. Research Policy, 32, 923 - 943.

Lakhani, K.R. and Wolf, R.G. (2005). *Why hackers do what they do: Understanding motivation and effort in free/open source software projects*. Retrieved on 13 March 2007 from http://opensource.mit.edu/papers/lakhaniwolf.pdf

Lerner, J. and Tirole, J. (2000). *The simple economics of open source* (NBER Working Paper No. 7600). Cambridge, MA: National Bureau of Economic Research. Retrieved on 7 November 2006 from http://www.nber.org/papers/w7600

Lerner, J. and Tirole, J. (2002). *The scope of open source licensing* (NBER Working Paper No. 9363). Cambridge, MA: National Bureau of Economic Research. Retrieved on 7 March 2007 from http://www.nber.org/papers/w9363

Lessig, L. (2002). Open source baselines: compared to what? In: Hahn (Ed.) *Government preferences for open source* (50 - 68). Washington, D.C.: AEI-Brookings Joint Center for Regulatory Studies.

Metiu, A. and Kogut, B. (2001). *Distributed knowledge and the global organization of software development*. Retrieved on 12 April 2007 from http://opensource.mit.edu/papers/kogut1.pdf

Microsoft (2005). *Government security program*. Retrieved on 11 April 2007 from http://www.microsoft.com/resources/sharedsource/Licensing/GSP.mspx

Microsoft (2007). *Compare security*. Retrieved on 13 September 2007 from http://www.microsoft.com/windowsserver/compare/linux/security.mspx

OSI (2007). *The open source definition*. Retrieved on 20 March 2007 from http://www.opensource.org/docs/definition.php

OSOSS (2007). *Open gemeenten. Resultaten van grootschalige inventarisatie inclusief voorbeeldprojecten*. Den Haag: OSOSS.

Paulson, J.W., Succi, G, and Eberlein, A. (2004). *An empirical study of open-source and closed-source software products*. IEEE Transactions on software engineering, 30 (4), 246 – 256.

Perens, B. (2005). *The emerging economic paradigm of open source*. First Monday, Special Issue #2.

Phillips, D. (2004). *The software project manager's handbook* (2nd edition). Hoboken, NJ: John Wiley & Sons.

Preece, J. (2000). *Online communities. Designing usability, supporting sociability*. Chichester: John Wilet & Sons.

Raymond, E.S. (2000). *The cathedral and the bazaar*. Retrieved on 9 October 2006 from http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/

Roberts, J. (2000). *From know-how to show-how? Questioning the role of information and communication technologies in knowledge transfer*. Technology Analysis and Strategic Management, 12 (4), 429 - 443.

Sadowski, B.M. and Rasters, G. (2005). *The end of communities in open source projects? Evidence from the Debian case* (ECIS Working paper [05.09]). Eindhoven: Ecis. Retrieved on 12 October 2006 from http://fp.tm.tue.nl/ecis/

Schmitz, P.E. and Castiaux, S. (2002). *Pooling open source software. An IDA feasibility study*. Brussels: European Commission, DG Enterprise.

Schmitz, P.E., *et al*. (2004). *Guideline for public administrations on partnering with free software developers*. Brussels: European Commission, DG Enterprise.

Sharma, S. *et al*. (2002). *A framework for creating hybrid-open source software communities*. Information System Journal, 12, 7 - 25.

Simon, K.D. (2005). *The value of open standards and open-source software in government environments*. IBM System Journal, 44 (2), 227 - 238.

Van der Pols, R. (2001). *ASL. Een framework voor applicatiebeheer*. Den Haag: ten Hagen Stam.

Van Wendel de Joode, R. (2005). *Understanding open source communities - An organizational perspective*. Delft: Technische Universiteit Delft.

Vendrik, C.M.M. (2002). *Motie van het lid Vendrik C.S*. Den Haag: Tweede Kamer der Staten-Generaal.

Waring, T. and Maddocks, P. (2005). *Open source software implementation in the UK public sector: evidence from the field and implications for the future*. International Journal of Information Management, 25 (5), 411 - 428.

Warton, R. *et al*. (2005). *Governance and global communities*. Journal of International Management, 11 (2), 125 - 142.

Wernberg-Tougaard, C. (2006). *What about support? Towards a new support economy?* Retrieved on 6 March 2007 from http://www.publicsectoross.info/images/resources/11_933_file.pdf

Wernberg-Tougaard, C., *et al*. (2007). Evaluating open source in government: methodological considerations in strategizing the use of open source in the public sector. In: Lytras, M. and Naeve, A. (Eds.) *Open source for knowledge and learning management. Strategies beyond tools (175 - 218)*. Hershey, PA: Idea Group Publishing.

Appendix F Interviews

F.1 A-select

Interview Bart Kerver
1 juni 2007

Korte beschrijving van het project
Vijf jaar geleden zijn wij gestart met wat uiteindelijk A-select is geworden. Vanuit Surfnet doen we voor het hoger onderwijs een aantal innovatietrajecten. Een van de dingen waar we altijd naar hebben gekeken is middelware. En dan in de breedste zin van het woord, van DNS, tijdsevices, en autorisatieservices. Een van de pilots die we deden met een externe partij was een applicatie voor studentenverkiezingen op een universiteit. Die werkte prima, maar toen kwam er een andere universiteit die daar ook wel belangstelling voor had, maar hun authenticatiemechanisme was net iets anders. Dat was dus de geboorte van A-select. De manier van authenticatie, een wachtwoord, een chip, biomedische data, werd losgekoppeld van de applicatie. Een tijd lang hebben we dat ontwikkeld samen met Alfa & Arris. Daarbij hadden we de verdeling gemaakt dat het closed source zou zijn, waarbij zij de rechten hadden voor niet-onderwijsinstellingen, en wij het beschikbaar zouden krijgen voor onderwijsinstellingen. De reden was dat we heel lang vanuit het onderwijs hebben geïnvesteerd in ontwikkelingen van middleware. Dat kostte veel, en bij dit project hadden we co-funding. Zij zouden dan ook een deel betalen.
Dat product is zich in de loop van de tijd – mede door allerlei mensen die er veel in het veld over vertelde – bij de overheid terecht gekomen. Die zijn daar ook een aantal pilots mee gestart. Toen was het nog steeds closed source. Een belangrijk moment was in juni 2004. Toen was de penetratie van A-select binnen het onderwijs zo hoog, en binnen de bibliotheken ook hoog, en een initieel aantal pilots binnen de overheden, dat men zei: het is vervelend dat we geen echte zekerheid hebben over de toekomst van A-select, het zou eigenlijk open source moeten zijn. Toen hebben we vanuit het onderwijs er voor gekozen om de bestaande source code af te kopen. Daarmee werden wij eigenaar en hebben we het open source gemaakt. Er zijn financiële participaties geweest van uit Surfnet, Kennisnet en een aantal andere partijen. Toen kwam de vraag wat voor open source licentie moet dat dan zijn. Omdat het product toen nogal nieuw was, wisten we niet echt welke licentie daarvoor geschikt was. We hebben daarvoor een externe deskundige ingehuurd, en ook OSOSS heeft er naar gekeken, en daar is een BSD-like licentie uitgekomen. Maar de licentie is niet helemaal conform een bestaande licentie. De reden daarvoor is dat we wilde dat mensen het product konden oppakken, er een willekeurig product mee konden maken en het niet per se hoefde terug te voeren naar de bron. En een andere reden was de oorspronkelijke ontwikkelaars met naam en toenaam genoemd moesten blijven worden in de licentie. Of dat nou een gelukkige keuze is geweest is niet helemaal duidelijk. De code wordt door allerlei partijen opgepakt, dat is positief, maar je kunt een fork beginnen en hoeft dat niet terug te voeren, waardoor we niet altijd zicht hebben op wat er nou precies gebeurt. Dat is jammer voor het project, aan de andere kant, de grote spelers voeren wel alles netjes terug. Achteraf gezien had een andere vorm van licentie het product misschien sneller doet ontwikkelen.

Product helemaal nieuw ontwikkelt, of gebaseerd op bestaande software
Het is helemaal nieuw ontwikkeld. 2001 was de formele start. Maar de eerste pilots waren al begonnen in 1999. Op het moment dat wij begonnen was er nog helemaal niets. Je ziet nu wel dat er vergelijkbare producten op de markt komen van commerciële vendors. Onze code is vanaf regel 0 helemaal zelf geschreven. En naarmate er meer organisaties bij kwamen heb ik geprobeerd sturing te geven aan het product.

Extern ontwikkelt, waarom?
Dat doen we eigenlijk altijd. Wij doen alleen maar tactisch of strategisch management.

Hoe vinden partijen elkaar?
Al heel snel hebben we er voor gezorgd dat er rapportages kwamen over wat wij ontwikkelden. Dat is eindelijk standaard voor al het innovatiewerk dat wij doen. Toen er een product leek te ontstaan hebben we direct alle domeinen geregistreerd en er een website aangehangen. Daarnaast hebben we heel actief pilots gedraaid met allerlei onderwijsinstellingen. Ook hebben we een belangrijke taak om onze kennis te dissimileren. Daarvoor hebben we hier een marketingapparaat en vanuit de Stichting Surf zijn er mensen die zich er heel erg voor inzetten. Verder veel mond-op-mond reclame en spreken op seminars en dergelijke. De belangrijkste doelgroep was dus wel het onderwijs, maar we zagen al heel snel dat er veel meer sectoren interesse hadden. Dus alle niet-commerciële partijen hebben we er ook een beetje bij betrokken. We vinden het toch ook belangrijk dat commerciële partijen het gebruiken, allen grijpen die partijen toch vaak terug op off-the-shelf software. Maar gelukkig komen we wel steeds meer commerciële partijen. We hebben die dus niet actief buiten de deur gehouden, maar ons er ook niet echt op gericht. Van oudsher zitten er dus al banken bij, en een digitaal notariskantoor.
Er is binnen Nederland een vrij kleine club die zich bezig houdt met identity management, en die kennen elkaar ook vaak goed en weten wat er speelt in de markt. Dus dan raakt een product al vrij snel bij iedereen bekent.

Structuur van het project
Nadat we de code hadden gekocht moesten we ook borgen dat de ontwikkeling goed was geregeld. We hebben toen een raad van advies opgericht. Daar zitten afgevaardigde in van alle grote stakeholders. Heel lang is ook de betaling vanuit Surfnet gedaan. Het grootste deel wordt nu ook nog vanuit Surfnet betaald. Dat willen we wel graag anders regelen. Donaties aan het project kunnen op verschillende manieren geregeld worden: je kunt testen, kennis inbrengen, en je kunt door externen dingen laten maken en die terug doneren. En dan zorgen wij er voor dat dat wordt geïntegreerd in een release. Dat is ook al een aantal keren voorgekomen. Over een aantal weken komen de technische mensen uit de groep ook bij elkaar en dan bepalen we samen waar we naar toe willen. Vanuit Surfnet geven we dan wel een voorzet op die roadmap.
Er is een lange tijd geweest dat we dat wilden formaliseren door er een stichting van te maken. Vanuit OSOSS is er toen ook mee gedacht over hoe je een community vorm zou moeten geven. En dat is eigenlijk het Apache model. Met veel verschillende lagen. Dat advies ligt er al heel lang, maar er is nooit wat mee gedaan. En daarom lopen we wel tegen een probleem aan. Bij een nieuwe release kunnen we of alles zelf betalen, of heel veel gaan lubben om de donaties te krijgen van anderen. Een nieuwe release kost tussen de 50.000 en 100.000 euro, dus dat is best wel veel geld voor ons. We hebben bij Surfnet geld voor innovatie, en dit ligt nu heel erg aan tegen productie. Was er een stichting geweest dan was het veel makkelijker op te zeggen dat bepaalde partijen gaan betalen als er een nieuwe release aankomt.
Nu komen alle bug fixes en nieuwe features nog bij mij terecht. Dat vind ik verder prima, maar het is wel raar. Het zou veel logischer zijn als ook de overheid, in de vorm van bijvoorbeeld GBO.overheid, er wat aan zou doen en er zeggenschap over zou hebben.
Ad-hoc of planning vooraf
Er zijn twee dingen. Bij het open source maken van A-select zijn wij gestart met een roadmap voor de ontwikkeling van A-select. Dit is de laatste jaren heel erg technology push en op basis van feature requests geweest. De raodmap geeft vooral globale ontwikkelingen.

Belangrijkste partijen die programmeren
Tot nu toe was het zo dat wij bijna alles bedachten. Vooral omdat wij heel veel contacten hebben in de wereld van identity management, en dus weten wat er nodig is in zo'n product. Inmiddels is het product zo volwassen dat ook marktpartijen het oppakken. Van oudsher hebben we altijd Alfa & Arris de opdracht gegeven als wij iets wilde laten maken. We wilde het steeds zo efficiënt mogelijk laten doen, en zij kennen de code goed. Maar doordat ze zo bekend zijn geworden hebben ze wel heel veel werk gekregen. Dat is positief voor hen, maar ze zitten daardoor wel veel drukker. Daarom kijken we ook naar alternatieve partijen. Gedurende twee jaar proberen we al andere partijen te zoeken, en er zijn nu drie a vier andere partijen waar je met een opdracht naar toe kunt gaan. Grote dingen besteden we nu ook bij meerdere partijen aan, en dan zie je vrij snel of zo'n partij veel van het product af weet, of juist niet. Als ze er weinig van af weten zijn de kosten heel hoog en de beschrijvingen vaak erg vaag. De laatste twee structurele ontwikkelingen zijn ook niet door Alfa & Arris gedaan, maar door een andere partij.

Review op de code
Vanoudsher deed Alfa & Arris dat ook, ook als het door een ander bedrijf is geschreven. Maar dat gaat nu een beetje een probleem opleveren omdat er steeds meer donaties komen van buitenaf. Ik weet nu dat ICTU een donatie gaat doen voor een structurele ontwikkeling. En het is zo dat als het ene bedrijf het maakt, en het andere bedrijf kijkt er naar, dan komt er altijd commentaar. Daar is wel een oplossing voor, dan moet je een coderingsstandaard definiëren. Op dit moment zeggen we alleen dat je een professionele manier van werken moet hebben, en de gebruikelijke manier van documenteren moet aanhouden. Als we echt met zo'n standaard zouden werken dat worden de ontwikkelkosten niet te betalen.

Financiële zaken
Er wordt geen geld gedoneerd, maar alleen functies. Ik had wel gehoopt dat als we een stichting zouden hebben dat elke partij dan naar ratio een bedrag voor de doorontwikkeling zou gaan betalen.

Community
Die is er zeker. De commerciële community is er, die verdienen een boterham aan het product. Die moet er ook zijn, en die ondersteunen wij ook. Daarnaast is er een groep die betaald en onbetaald support leveren op de mailinglijst. Dat zijn soms de programmeurs, maak af en toe ook mensen van MMBase en van Cap Gemini. Er komen niet heel veel vragen op de mailinglijst, maar als ze komen zijn ze meestal wel binnen een paar uur opgelost. Dat er niet heel veel vragen komen, komt omdat alle partijen die A-select gebruiken ook gebruik maken van een systeemintegratiepartij. Zij willen niet zelf zitten modderen met de code. En die partijen hebben wel weer alle kennis van A-select, dus die hoeven nauwelijks iets te vragen.

Relatie met DigiD
DigiD kent twee smaken: voor bedrijven voor burgers. Die voor bedrijven is het oudst. Ze hebben toen – heel onverstandig – een fork gemaakt van A-select 1.3, nog net voordat het open source werd. DigiD voor burgers draait een gewone productierelease van A-select. Ik hoor wat geruchten dat ook DigiD voor bedrijven overgaat naar de gewone A-select. De fork zal dus wel op den duur uitsterven.

Snelheid van ontwikkeling
We proberen om jaarlijks een release te maken. Alleen echte security issues brengen we tussentijds als patches uit. Zou je het veel vaker doen, dan kunnen onderwijs en DigiD het niet bijbenen.

Nederlandse organisaties?
Ja, maar we hebben wel intensieve banden met Europese en Amerikaanse organisaties. Sommige van die organisaties gebruiken A-select wel op zeer kleine schaal. Voor bijvoorbeeld buitenlandse onderwijsinstellingen zul je zien dat ze heel goed kijken naar het lokale draagvlak voor een systeem. Als daar niet van A-select gebruikt wordt gemaakt, zullen andere het ook niet snel doen. Ik vermoed dat je A-select voornamelijk in Nederland zult zien vanwege de penetratie bij de overheid. Wat je wel ziet is dat we veel minder naar producten kijken, maar meer naar de grote standaarden. Sinds versie 0.2 hebben wie iets dat heet Cross A-select, daar kun je een aantal instellingen aan elkaar koppelen. Later is er een standaard ontstaan, gebaseerd op XML, en dat hebben de toegevoegd aan A-select. En dan maakt het op den duur niet meer uit of je connect met A-select of met een ander product, als ze allemaal die standaard kunnen lezen.

Is taal ook een probleem?
Nee, alles is in het Engels. Dus daar ligt het niet aan. Men kijkt alleen naar nationaal niveau. Maar we wilden alles in het Engels doen omdat we internationale uitwisselingen hebben, en we dan ook internationaal kunnen gaan.

Hoe is ondersteuning geregeld?
In de basis doen wij geen ondersteuning meer. Vroeger deden we op basis van mixed funding de implementatie van A-select bij onderwijsinstellingen. Dat deden we onder twee voorwaarden: de instelling betaald de helft, en het moet een productieomgeving opleveren. Zo is het gebruikt heel erg gestimuleerd. Daarna was A-select vrij volwassen, en moest de markt het oppakken. Wel leveren wij nog beperkt technisch support. In het geval zaken niet door de mailinglijst worden opgelost betalen wij Alfa & Arris om het op te lossen.

Willen jullie er op den duur uitstappen?
Wij zijn ook gebruiker van het product, dus blijven we er ook bij betrokken. Ik zie ook nog niet zo snel dat een universiteit zelf gaat ontwikkelen. Wij willen niet alles zelf doen, maar zolang niemand anders het overneemt blijven we het wel doen.

Aan welke ondersteuning is behoefte?
Vooral als het om hele specifieke technische ondersteuning gaat.

Waarom open source?
Het was vooral de vraag van de markt. Vooral een universiteit wilde een vendor lock-in voorkomen.

Voor en nadelen van open source
Ik denk niet dat er nadelen zijn geweest. Behalve misschien het licentiemodel dat we hebben gekozen. Verder denk ik dat het in het voordeel van het product heeft gewerkt. Maar er zijn misschien wel instellingen geweest die open source eng vonden. Alfa & Arris is nu ook bezig met een fork van A-select, dat een commerciële versie van A-select wordt. Ik verwacht dat daar ook veel belangstelling voor is. Alfa & Arris is bijvoorbeeld ook verantwoordelijk voor DigiD burger, dus misschien gaan zij ook wel de commerciële versie gebruiken.

Open source en de overheid
Heb ik weinig zicht op.

Verwachtingen van dit onderzoek
Het voorstel voor het Apache model was heel helder. Maar het probleem was dat de partijen niet bij elkaar zijn gekomen. Eerst waren ze het er mee eens, maar toen de juristen er naar gingen kijken waren er opeens allerlei problemen. En het zonder een stichting regelen is heel complex. We hebben ook lange tijd gekeken naar MMBase. Maar ook MMBase loopt nog niet heel goed. Ik ben dus benieuwd waarom het bij Apache wel werkt, maar in Nederlands niet echt. Ik vermoed dat het te maken heeft met de omvang het product. Het heeft veel te maken met kritische massa, en ik zou niet weten hoe je die kunt veranderen.

Other sources used:
- A-select website. http://a-select.surfnet.nl/
- A-select website. http://aaa.surfnet.nl/info/a-select/home.jsp

F.2 APLAWS+

Sources used:
- Personal conversation with Tracy Wang, 28 November 2006, Brussels

- APLAWS+ website. http://www.aplaws.org.uk/
- Public sector and OSS website http://www.publicsectoross.info/news/newsdetail.php?Id_news=5

F.3 eFormulieren

Interview Pim Schouten
14 mei 2007

Korte uitleg over het project
Het programma eFormulieren is een programma dat een bouwsteen van de e-overheid aan het inrichten is waar allerlei overheidspartijen hun digitale dienstverlening naar burgers en bedrijven mee kunnen inrichten. Wij zorgen niet alleen voor de voorziening van formulieren, maar ook het ontwikkelen en in productie brengen van die formulieren. Als een gemeente zegt dat ze een aantal diensten hebben waarvoor de burger naar het gemeentehuis moet komen, dan kunnen wij een heel groot deel van die diensten digitaal maken.
Toen het project startte wilde we alles centraal hosten. Nu zitten we een beetje op een omslagpunt daarmee. Nu stellen we de applicatie ook ter beschikking aan overheidspartijen zodat ze het zelf op hun eigen omgeving kunnen draaien. Zover zijn we echter nog niet, want daarvoor moet nog het een en ander in de software gebeuren.

Op basis van bestaande software
We hebben een Europese aanbesteding doorlopen. Die is gewonnen door Logica CMG met ePlatform. Dat is een applicatie die zij een tijdje daarvoor in een open source domein hadden ondergebracht. Daarop hebben wij doorontwikkeld.
Er zijn eigenlijk vier componenten. De eerste is de eFormulieren voorziening, dus de applicatie die het formulier zichtbaar maakt. Dan is er de message broker, die berichten tussen overheidspartijen verstuurt. Aan de achterkant zit de work bench – die ook weer gesplitst kan worden in een library, waarmee je formulieren kunt maken, en een beheeromgeving.

Zelf ontwikkeld of uitbesteed
We hebben alles uitbesteed aan Logica CMG, maar we willen we nu van af omdat we niet afhankelijk willen zijn van één leverancier. Logica CMG heeft op dit moment namelijk een capaciteitsprobleem. Partijen als Atos Origin of Cap Gemini zouden we graag mee laten ontwikkelen. Dus nog even niet de applicatie helemaal beschikbaar stellen in het open source domein (wat die overigens wel is), maar de doorontwikkeling wel in eigen hand houden. Wij willen wel uiteindelijk de specificaties voor doorontwikkeling schrijven, en dat dan uitbesteden aan andere partijen.
Het probleem is nu dat als we wat nieuws willen we dit bij Logica CMG kunnen neerleggen, of bij een andere partij die er minder kennis van heeft, maar dat zal altijd langer duren.

Hoe vinden partijen elkaar?
We hebben een aantal accountmanagers, die partijen gaan benaderen. We hebben voor decentrale overheden een aantal standaard formulieren. Samen met onder meer EGEM hebben we die ontwikkeld, zodat we het niet voor elke gemeente overnieuw moeten doen. Voor andere instanties maken we unieke formulieren. Bij UWV maken ze de formulieren zelf, maar voor andere grote partijen maken wij de formulieren.

Distributie naar andere partijen
Op dit moment is er één implementatie van ePlatform. Die wordt gehost bij Logica CMG. Als er nieuwe functionaliteit komt dat is die meteen voor iedereen beschikbaar. Als het decentraal gaat draaien dan zorgen we dat we op een plek de workbench houden. Maar iedereen krijgt dan zelf een library om formulieren te maken. Dat betekend dat we niet tegelijkertijd heel veel versies moeten ondersteunen.

Organisatie van het project
Wij hebben een gebruikersgroep. Daarin zitten een aantal belangrijke gebruikers. Onze ideeën worden daar getoetst, en we komen ook zelf met ideeën. Die groep komt een keer in de maand bij elkaar. In de toekomst hopen we dat die partijen ook zelf dingen gaan bouwen, of laten bouwen.

Financiën
In principe betalen wij alles. Aan kosten voor ontwikkeling en hosting betalen gemeentes niks. Alleen als er heel specifieke wensen zijn die niet inzetbaar zijn voor anderen, dan moeten ze dat betalen. Bij twee ministeries hebben we een koppeling moeten laten ontwikkelen tussen ons systeem en hun backoffice. Gemeentes hebben dus ook niet rechtstreeks te maken met Logica CMG.

Communities
Het is onze taak om dat vorm te geven. Dat is iets dat moet gaan leven. Je hebt de gebruikers en de ontwikkelaar. De gebruikerskant begint nu vorm te krijgen, partijen komen met eigen ideeën. Aan de ontwikkelkant is er nog maar een partij, maar dat moeten er meer worden. Maar dat is wel veel wat je aan een partij vraagt. Zo'n partij moet namelijk eerst heel veel tijd steken in het leren kennen van de code voordat ze er succesvol op kunnen doorontwikkelen. Het is nog de vraag wie dat allemaal gaat betalen, en of wij dus actief opdrachten gaan aanbieden aan andere partijen. Maar er zijn in ieder geval wel een aantal partijen die geïnteresseerd zijn. En voor die bedrijven zijn er waarschijnlijk ook genoeg opdrachten.
De meeste overheden hebben geen echte behoefte om lokaal dingen te ontwikkelen. Die vinden het al lang prima dat wij dat doen. Voor hen is het dus ook prima als de hosting centraal blijft. Voor partijen als het UWV ligt dat anders. Zij hebben heel veel formulieren die heel veel gebruikt worden. Het kan zomaar zijn dat de wetgeving veranderd, en een dag later moet dan het formulier veranderd zijn. Zij kunnen er dus niet op wachten tot een ander het doen, maar doen het zelf.

Wat krijg je van gebruikers terug?
Partijen zullen nu zelf moeten gaan doorontwikkelen. Formulieren kunnen nu worden aangeleverd als een PDF bij een e-mail, of als een XML bericht dat direct ingelezen kan worden in het back office systeem. Dat tweede is best ingewikkeld, want de gemeente moet hun eigen systemen aanpassen.

Willen gemeentes lid worden van een community?
Grote gemeentes wel, maar kleine gemeentes niet. Soms hebben ze een wens en roepen ze iets, maar ze zullen geen actieve rol gaan spelen.

Quality checks op code van andere
Dat is wel iets dat moet gaan gebeuren, want we willen alleen goede code, als die code direct gebruikt wordt door heel veel partijen. We hebben alleen nog geen idee hoe we dat precies gaan doen. Ik zou me kunnen voorstellen dat Logica CMG daar een grote rol in gaat spelen. maar dan maken we ons toch weer afhankelijk van een partij, en dat is juist iets dat we niet willen.
Wat er moet gebeuren is dat de code veel modulairder moet worden opgebouwd. Een partij is dan eigenaar van een module en kan die onderhouden. Er komt dan nog steeds wel een rol te liggen bij Logica CMG, maar deze is veel kleiner, dus dat is minder erg. Het zou ook kunnen dat het in de toekomst bij GBO belegd gaat worden, niet alleen voor eFormulieren, maar voor meer programma's. ICTU heeft gewoon niet de capaciteit om dat te doen.

Wil Logica CMG het ook dat andere partijen mee gaan doen?
Ze zitten er een beetje duaal in. Aan de ene kant zijn ze er blij mee, want dat neemt de druk op hun weg, en zij kunnen gebruik maken van dingen die door anderen zijn geschreven. Aan de andere kant is dat wel omzet die daar weg gaat. Zij hebben de keuze gemaakt om het open source te maken, dus dan moeten ze daar ook achter staan. Maar in de praktijk is het altijd een beetje lastig.

Ondersteuning op de code zelf
De code staat op SourceForge en het uitwisselplatform. Alle documentatie is ook openbaar. Wij onderhouden de code niet omdat we de competentie er niet voor hebben. Als er problemen zijn dan geven wij die door aan Logica CMG. We bekijken wel alles wat binnen komt, bundelen verzoeken, en doen het uiteindelijke release management.

Wat voor ondersteuning is behoefte aan
Wij zijn de primaire ingang en de gemeente. Er is geen contract tussen Logica CMG en de gemeente. Dus ze moeten bij ons komen. Het kan zijn dat het gaat om een wijziging in een formulier – daar hebben we Logica CMG niet voor nodig – of kan gaan om een wijziging in de code. Dan prioriteren we alles wat binnenkomt en geven het door. Dat dingen niet goed werken gebeurt bijna nooit, maar er zijn wel heel veel wensen. Die voeren we wel allemaal uit.

Bewuste keuze voor open source
Niet helemaal. Je kunt open source heel moeilijk eisen in een aanbesteding. Maar diegene met open source had wel een groot voordeel bij de aanbesteding, omdat juist het open source is.

Licentie
GPL, dat was al zo, dus daar zijn we mee verder gegaan.

Voordelen van open source
Het grote voordeel is dat meerdere partijen op de code kunnen gaan ontwikkelen, zowel ICTU als bijvoorbeeld UWV. Een nadeel is er niet echt, maar wel een risico. Dat is om alles bij elkaar te brengen en te houden en eens in de zoveel tijd alles te bundelen tot een nieuwe versie.

Speciale tools
De code staat wel op SourceForge, en daar blijft het ook staan, maar het echte ontwikkelen zal toch plaatsvinden bij grote partijen. Het zal geen applicatie worden waar hobbyisten mee aan de slag zullen gaan. We zullen dus ook alleen releases op SourceForge zetten die af zijn, en niet de ontwikkelversies. Het kan namelijk zo zijn dat de code bekeken gaat worden door een partij waarmee we niet in gesprek zijn en dan de code gaat gebruiken. Dat kan, en dat mag, maar daarom willen we er alleen echt goed geteste versies hebben staan. Hoe we dan precies coördinatie tussen de ontwikkelaars zullen gaan regelen, dat weten we nog niet. Want waarschijnlijk hebben de wijzigingen van de ene partij ook wat gevolgen voor de andere partij. We willen dit voorzichtig opbouwen en andere partijen eerst laten ontwikkelen aan de randen van de applicatie, dus koppelingen met andere databases of zo. Dan worden die bedrijven meer competent en kunnen zo ook de ingewikkeldere zaken aan.

Genoeg stimulatie voor open source
Dan moet je denk ik een onderscheid maken tussen nieuwe omgevingen en vervanging van bestaande omgevingen. Er wordt nog veel gesproken over open source, maar het wordt heel weinig toegepast. Daar waar het wordt toegepast is vaak in omgevingen als deze, waar niks was, een aanbesteding is geweest, en open source uitkomt. Bij vervanging zie je wel heel veel initiatieven, maar heel weinig realisatie. De meest makkelijke applicaties zijn kantoorapplicaties. Bijna elke gemeente praat wel over OpenOffice, maar alle initiatieven stranden omdat ze niet afhankelijk willen zijn van de hobbyist op de zolderkamer en omdat je te maken hebt met hoge migratiekosten.
Ook bij het verkopen aan gemeentes zijn we aangelopen tegen het open source image, dat we ze echt moeten overtuigen dat het geen hobbyistenproduct is, maar dat wij de ontwikkeling doen.

Is de overheid verplicht open source te gebruiken?
Nee, geen verplichting. Maar wel een sterke wens. Juist geen verplichting vanwege de regels van de aanbesteding. Maar je kunt wel een sterke voorkeur uitspreken. De overheid zou in ieder geval een sterke stimulator moeten zijn.

Verwachtingen ten aanzien van het onderzoek
Er is veel theoretisch kennis aanwezig. Maar de kennis van het bouwen van een community in de praktijk ontbreekt. Dat is nog maar heel weinig gedaan. En dat zou ik heel graag willen weten.

F.4 Flamingo

Interview Ron Wardenier
12 maart 2007

Korte beschrijving van het project
Flamingo is een kaart viewer. Dit is een applicatie binnen een browser voor thematische kaarten. Je kunt inzoomen en informatie van objecten opvragen. Daarmee is het dus anders dan een papieren kaart. Deze is ontstaan uit de behoefte voor een risicokaart-viewer voor de provincies. Alle twaalf provincies gebruiken die nu, want het gebruik is verplicht. Hierdoor ziet de kaart er in elke provincie dus ook hetzelfde uit. Inmiddels kunnen er ook vele andere zaken op de kaart worden getoond. Dit is ook nodig omdat het werkterrein van de provincie breed is. Er was al heel snel veel belangstelling voor Flamingo, ook van buiten de provincie. We wisten toen niet zeker wie we wel en niet toegang zouden geven, en hoe we dat dan moesten ondersteunen. Zo is de truc naar open source gekomen.

Door wie ontworpen
We hebben bijna alles zelf ontworpen. Maar je hebt een voorkant en een achterkant, de database. Dit is op dit moment commerciële software en heeft dus weinig te maken met Flamingo. Maar echt alleen de viewer is door ons ontwikkeld. Alleen een script voor flash-player detectie is niet door ons ontwikkeld, maar is wel open source.

Waarom zelf ontwikkelen
Per provincie is het verschillend. Groningen heeft geen ontwikkelaars. Friesland heeft dat wel. Ze hadden al een risicokaart, maar die vereiste heel veel werk om aan te passen. Dat was dus niet handig voor ze. Friesland wilde een betere kaart, en dat wilde ze zelf doen, zodat ze het beter in de hand kunnen houden. Het is dan meer een organisch proces, wat leidt tot beter resultaat. Ook het InterProvinciaal Overleg stimuleerde zelf ontwikkelen. Alleen als je het niet zelf kunt moet je het uitbesteden. Dus omdat Friesland het toch al deed kon het ook makkelijk gedeeld worden.
In het begin was het nog niet duidelijk dat het open source zou worden. Toen was het nog iets dat gewoon door de provincies werd gedeeld.

Gebruik gemaakt van andere componenten
Alleen de detectie voor de flash-player.

Hoe hebben partijen elkaar gevonden
Provincies kijken bij elkaar, en nemen met elkaar contact op als ze iets leuks bij een ander zien. Het verspreiden ging dus vrij vanzelf, we hebben er geen reclame voor gemaakt. En al vrij snel toonden ook Rijkswaterstaat, waterschappen en gemeentes belangstelling.

Leiding over het project
Er is een Flamingo beheer groep. Hierin zitten de acht provincies die er vanaf het begin bij betrokken waren. Ikzelf zet die groep voor. De vier andere provincies gebruiken Flamingo wel, maar zitten niet in de beheergroep. Deze groep bepaald als enige wat Flamingo wel en niet kan. We zijn nu bezig met Flamingo 2, die niet meer kan, maar wel helemaal is opgesplitst in modules.

Rollen van de verschillende gebruikers
Die zijn niet heel duidelijk afgebakend. Het is niet heel formeel. Organisatorisch hangt het onder InterProvinciaal Overleg GEO. Het is niet echt een projectgroep, want die zijn eindig. Het is de bedoeling dat het voorzitterschap om de twee jaar gaat wisselen. De enige ontwikkelaar is Menko Kroeske, van Friesland. De overige deelnemers zijn werkzaam bij provincies in verschillende rollen. Hun rol bestaat vooral uit testen en meebepalen wat er wel en niet in Flamingo komt. Tot aan versie 1 was er een persoon die ook de documentatie schreef, anders dan de programmeur. Nu doet Menko dat ook direct in de broncode. Wel wordt er door mij

nog af en toe extra documentatie geschreven waarom iets zo gedaan wordt en hoe je het kunt gebruiken. Op dit moment is de tijd die mensen er aan kunnen besteden nog te weinig, en dat is best een probleem. Ik probeer het af en toe in mijn vrije tijd, of op mijn werk, om de website bij te houden, maar dat lukt ook niet altijd.

Manier van werken
Er wordt niet gebruik gemaakt van speciale ontwikkeltools. Dat is vooral omdat er geen kennis is van dat soort tools. Het voltooien van versie 2 heeft nu de hoogste prioriteit. Uiteindelijk zullen we wel naar zo iets toe moeten, maar daar zullen we wel hulp bij nodig hebben. We zijn geen ICT'ers.

Hoe gaat het dan bij releases
Bij versie 1 hebben we het gebruik beperkt. Dus dan was het gewoon een mailtje sturen. Nu is het open source, maar aan versie 1 is toen niks meer veranderd, omdat we bezig zijn met versie 2. Hiermee zitten we nu in beta-stadium. Deze staat nu al op de website. Die wordt al gedownload, maar het zijn vooral de oudgediende die het weten te vinden. Als versie 2 er helemaal is dan moet er een press release komen die wereldwijd moet gaan.
Flamingo is een map viewer. Daardoor is het simpel. Dat is zijn grote kracht, maar ook een gevaar. Als beheer groep moeten we er voor zorgen dan Flamingo eenvoudig blijft, en dat er niet zo veel aan toe wordt gevoegd dat het een compleet GIS systeem wordt. Maar het blijkt wel lastig te zijn om zulke functionaliteit buiten de deur houden. Bijvoorbeeld, mensen willen een pijl er op tekenen, of een cirkel. Het is dan geen tool om te raadplegen, maar om te communiceren. En eigenlijk wijkt dat af van viewen. We gaan het toch ontwikkelen. Maar het blijven plugins. Die worden wel vanuit de beheergroep ontwikkeld, maar worden wel naast de bestaande Flamingo gezet. Sommige modules worden ook extern ontwikkeld. Zwolle en de provincie Utrecht hebben bijvoorbeeld extra dingen laten ontwikkelen door commerciële bedrijven. Die extra zaken worden ook allemaal open source. Maar er is een probleem als zo'n project eindigt. Wij kunnen het dan niet allemaal ondersteunen. Wij proberen dan een GIS systeem te voorkomen, maar andere zorgen er wel voor dat we zo'n compleet systeem krijgen, eigenlijk tegen onze wil in. Daarom komen deze modules er ook echt naast te staan.
In de beheergroep gaat het nu vooral over versie 2 en welke modules er moeten komen. In de groep gaat alles vrij democratisch. Iedereen is het er vrijwel over eens wat het moet zijn. Als er behoefte is aan meer dan zorgt een provincie daarvoor. Naar verloop van tijd kan er dan iets uit komen waar ook de rest wat aan heeft.

Hoe is het financieel geregeld
Het gaat allemaal met gesloten beurzen. Menko wordt door Friesland betaald, maar zij zeggen dat ze het toch hadden gedaan, dus dat het niks extra's kost. We hebben nu wel eProvincies om geld gevraagd, een soort beheerbudget. Er is bijvoorbeeld behoefte aan een printfunctie. Die is er niet maar moet er wel komen. We hebben dat budget wel gekregen, maar dan kom je weer in de hele molen van budgets en verantwoordelijkheden. Dat blijkt nu wel ingewikkeld te zijn. Maar we krijgen uiteindelijk 20.000 euro per jaar. Hierdoor kunnen we dan dingen laten bouwen. Dit zal vooral door externe, commerciële bureaus gaan gebeuren. We willen bijvoorbeeld ook een configuratiemodule hebben.

Speciale gebruikersgroepen
Nu zijn de mensen in de beheergroep de gebruikers. Dus er is verder geen gebruikersplatform. Maar als het project groeit kan het anders worden. Er is nu een community website. Hierin zitten forums en zo. Dat wordt dus de plek om alles te organiseren. Maar veel mensen mailen mij nu ook nog direct. Die verwijs ik eigenlijk altijd door naar het forum. Maar het blijft lastig, want dan moet het in het Engels, en iedereen kan dan meelezen. Ik ben tot nu toe ook diegene die de vragen beantwoord, maar dat zouden eigenlijk alle mensen van de beheergroep moeten doen, maar ik vrees het ergste... Maar ik hoop dus dat er een aantal enthousiastelingen zijn die zich er wel op willen storten.
We zijn ooit open source gegaan omdat we geen redenen konden bedenken om het bepaalde groepen te ontzeggen. Maar hierbij geven we wel minimale ondersteuning. Dat kun je ook niet echt van ons verwachten. Eigenlijk doen we al meer dan dat we strikt genomen zouden moeten doen. Iedereen mag dus eigenlijk blij zijn dat het er – gratis – is. Dan kan het twee kanten op gaan. Organisaties kunnen zeggen dat de basis te smal is om het te gaan gebruiken. Aan de andere kant, hoe meer organisaties het gaan gebruiken, hoe sterker het project komt te staan. En we kijken wel hoe ver we komen. In Frankrijk worden heel veel leuke dingen gedaan met Flash, dus wie weet. Maar het zou ook zomaar zo kunnen zijn dat het project over een jaar helemaal dood is.

Code accepteren van anderen
Dat is wel de bedoeling. Dan komen er gewoon heel veel modules. Mensen kunnen dingen uploaden, we kijken er even na, en dan wordt het vrijgegeven. Dit wordt dan gedaan door die beheergroep. En als het echt beter blijkt te zijn dan de standaard, dan zullen we het overnemen in de basisinstallatie. En als dat door iemand uit het buitenland wordt geschreven, zou het best zo kunnen zijn dat we hem vragen om plaats te nemen in de beheergroep. Dan moeten de vergaderingen alleen via de mail plaatsvinden, maar dat is niet zo'n probleem.
Het is denk ik wel waarschijnlijker dat het een extra component wordt en niet in de basis komt, omdat ik Flamingo klein en beheersbaar wil houden.

Hoe kunnen organisaties elkaar vinden
Flamingo staat al in een internationale catalogus. Maar bezoekers vanaf die site kwamen op een Nederlandstalige site, dus dat was niet zo handig. Nu het in het Engels is, moet dat wel beter gaan. En er komt natuurlijk een press release.

Bereidheid om die organisaties te helpen
Als er veel organisaties het gaan gebruiken neemt de behoefte aan ondersteuning natuurlijk ook toe. Bij mij, Menko, en twee of drie anderen van de beheergroep is er zeker de bereidheid om deze nieuwe gebruikers te gaan helpen. Probleem is dat niemand echt op zijn werk tijd heeft om er veel aan te doen, dus dan komt het er op neer, hoe leuk vind je het om het in je vrije tijd te doen.

Wat zou er gebeuren als een belangrijke ontwikkelaar er mee op houdt
Dat is niet het einde van het project. Er is wel genoeg kennis in Nederland, maar dan moet er echt veel geld bij. Het zou denk ik dan niet meer door een van de provincies gedaan kunnen worden.

Interesse commerciële bedrijven
In het begin was er een bedrijf dat dat wilde. Maar wij willen niet Flamingo een stempel geven van een bedrijf. Dus wij gaan niks aanmoedigen. Maar elk bedrijf staat vrij om het te doen. De belangstelling is er, maar iedereen kijkt de kat uit de boom. Maar bedrijven kunnen dus ondersteuning gaan aanbieden, als ze dat zouden willen.
Op het gebied van ondersteuning gaat er sowieso wat veranderen. GBO.Overheid wordt ook voor provincies, en gaat Flamingo in beheer nemen. Maar hoe dat uitgewerkt gaat worden weet ik nog niet. Ik heb dus geen idee wat GBO.Overheid precies voor beheer gaat doen, en voor wie. Maar het zal waarschijnlijk niet zijn dat zij vragen op het forum gaan beantwoorden, maar bijvoorbeeld wel contractbeheer als je een deel uitbesteed. GBO.Overheid gaat dus ook geen nieuwe dingen doen, alleen beheer. Dus er moet nog uitgezocht worden wie dan de ontwikkeling gaat doen.

Waarom open source
Het was eigenlijk een beetje een truc. Want als je het gewoon aan iedereen geeft, dan ben je daar verantwoordelijk voor, en verwachten ze ondersteuning. Maar we zijn geen softwarehuis. Dus open source was vooral een truc om het weg te geven, zonder verplicht te zijn ondersteuning te gegeven.

Welke licentie zit er op
GPL. Hierover is contact opgenomen met Bart Knubben. Deze was het meest passend.

Voordelen en nadelen van open source
Het project had er op dit moment niet anders uitgezien als het niet

open source was geweest. Maar er zijn nu veel bedrijven die het heel interessant vinden, en het goed volgen. Er is ook al een bedrijf dat een module in ontwikkeling heeft. Dat was nooit gelukt als het niet open source was. Ik weet nog niet welke kant het op zal gaan, maar volgens mij moet het wel aanslaan.

Open source en overheid: anders
Eigenlijk zou de overheid heel vaak met open source moeten werden. Het was destijds voor ons geen overweging. Op heel veel plaatsen binnen de overheid wordt het al wel gedaan. Voor bedrijven is dat goed, dat zorgt voor competitie. Binnen provincies, dat zijn er twaalf, doen ze bijna allemaal hetzelfde. Daar moet je dus samenwerken, anders maak je twaalf keer hetzelfde, zeker omdat het over belastinggeld gaat. Voor kleine gemeentes geldt dat ook, die kunnen veel gratis software krijgen. Als ze dan alsnog ondersteuning willen kunnen ze naar een commerciële partij. Voor bedrijven kan het wat minder zijn, want die hebben natuurlijk liever dat 400 gemeentes allemaal individueel producten van hen afnemen.

Grootste ervaring van open source maken
Tot nu toe niet echt veel. Maar het is wel mooi om te zien dat je veel kunt delen, en ervaringen uitwisselen met andere open source projecten. Maar als je kiest voor gesloten dan weet je zeker dat er niks op je af komt. En we merken ook dat mensen moeten wennen aan open source. Dat het geen risico is dat het over een jaar ophoudt, maar een kans dat je er altijd mee door kunt gaan. Ik denk ook niet dat er veel overheden zijn die voor open source kiezen als ze het niet kunnen ondersteunen. Of ze kunnen het en kiezen voor open source, of ze kunnen het niet en besteden het hele project uit, maar niet een project kiezen en ondersteuning uitbesteden.
Maar we maken ons ook niet al te veel illusies. De levensduur van Flamingo is niet zo lang. De ICT en GEO wereld veranderd zo snel dat het over een paar jaar al weer verouderd is. Zeker als bedrijven als Google en Microsoft er ook instappen, daar zit zo veel geld achter, daar kun je nauwelijks tegenop concurreren.

Other sources used:

- GEO-INFO. *Flamingo: de open source GUI*. 2006-09
- Flamingo website. http://www.flamingo-mc.org/

F.5 GemGids

Interview Holger Peters
20 april 2007

Algemene beschrijving van het project
GemGids komt voor uit EGEM, de gemeente Voorst, en Max.nl. Sinds 5 januari 2007 hebben we met GemGids een stichting opgericht. Voorst wilde eerst de basisadministratie op orde hebben, en daarnaast ook een visie op het hele IT beleid van de gemeente. Het doel was om efficiency te verbeteren. Via de browser konden dan allerlei applicaties benaderd worden. Het eerste doel was om dit vooral intern op orde te krijgen, maar daarna ook de slag naar buiten te maken, dus voor de burgers. Hierbij moest de burger kunnen meekijken in de systemen van de ambtenaren. Hiervoor hebben we toen EGEM benaderd. Daar heb ik een jaar lang gewerkt. EGEM wilde GemeenteUtopia starten, om te laten zie hoe je in een utopie zou kunnen samenwerken en alles zou kunnen delen. Daar is uitgekomen dat we samen het project Gemeente Pragmatika zouden opstarten. Het idee is dat als je je basisregistratie op orde hebt, en je hebt een productcatalogus, en die combineer je, dan je alle data kunt ophalen bij de verschillende organisaties, mits je open koppelvlakken hebt. Dus als je als sleutel een persoon, pand, of perceel hebt, en je stuurt die vraag naar een gemeente of provincie, dan moet die altijd informatie terug kunnen geven. Dat is het fundament geweest voor Pragmatika. Dat project is gedaan met vijf gemeentes, het Kadaster, en de provincie Utrecht. Voor dit project is toentertijd een aanbesteding uitgeschreven. Het bedrijf Max.nl heeft deze aanbesteding gewonnen. Die hebben toen een prototype gemaakt op basis van open source componenten, dus Linux, Apache, MySQL, PHP, Minnesota Mapserver. Dat functioneerde goed. Hierover hebben we ook veel verhalen gehouden bij andere organisaties, en dat sloeg erg aan. Maar het was meer een soort prototype voor ambtenaren. Er waren toen de koppelvlakken, en er kon een product komen die daarmee interfacete. Dat kon eigenlijk wel een “commercieel” product zijn. EGEM gaf echter aan dat zijn geen product op de markt wilde zetten. Toen is Voorst verder gaan kijken. Het probleem was dat je gewoon veel geld kwijt bent als je het goed wil doen, een paar miljoen, ook met open source componenten. Zoveel geld heeft Voorst niet, maar zo konden wel een eerste versie maken, met maar een paar producten er in, en dat dan beschikbaar stellen aan de markt. Maar dan loop je al direct tegen problemen aan als support en onderhoud.
We hebben die eerste versie gemaakt, vrij snel in twee maanden, maar daarna hebben we bijna een jaar er over gedaan om een heel stabiel framework te maken onder die code. Hierdoor konden weer nieuwe componenten snel gemaakt worden. Ondertussen had de gemeente Apeldoorn ook al een paar componenten gebouwd, en de provincie Gelderland ook. Er is vorig jaar al ongeveer 100.000 EUR in geïnvesteerd om een goede basisversie te krijgen.
Voor GemGids werken we onder andere met de API's van Google Maps en Google Eearth. Dat is dan niet open source, maar wel gratis. Voor EGEM hebben we ook een versie gemaakt met MapBuilder die wel open source is. We willen uiteindelijk niet te afhankelijk zijn van Google, maar dat is voorlopig wel mooier en duidelijker voor de klant.
Je kunt bij GemGids inloggen (via DigiD) en je krijgt op de kaart te zien waar je woont en kunt zien hoe je bij de overheid staat geregistreerd in de basisregistraties. Het is ook mogelijk om alle (bouw)vergunningen van de gemeente of provincie in je omgeving op te vragen. Ook kun je bijvoorbeeld op een huis klikken en de rioolaansluiting opvragen. Dat wordt dan weer uit een andere database gehaald. Alles gaat op basis van webservices, dat is ook het uitgangspunt, alles moet via standaarden, zoals die van de W3C.
Afgelopen januari hebben we een stichting opgericht die de code gaat beheren. Daar is geen geld voor. Eigenlijk zouden we willen dat iemand dat gratis gaat doen in een open source community, maar als je dat roept dan krijg je niet zo snel iemand. Als partner van de stichting GemGids heeft Max.nl/overheid als bedrijf hierin geïnvesteerd. We staan hiermee borg voor het support op de ontwikkelde software en de hosting (technische ondersteuning) van de diensten. Hiermee is het beheer geregeld. Als er een nieuwe versie van een browser komt dan passen wij het framework aan, zodat alle componenten die draaien op het framework blijven werken. Daarnaast zorgt Max.nl/overheid voorlopig voor de marketing, administratieve organisatie en huisvesting van de Stichting GemGids.

Wat is er ontwikkeld?
In de eerste plaats dus dat framework. En hierop kunnen applicaties worden ontwikkeld voor overheidsorganisaties als gemeenten, provincies en waterschappen. Dat is eigenlijk ook wat stichting GemGids doet, het is een community tussen bedrijfsleven en overheid. Dus als een overheid komt bij de stichting, dat kijken we welke componenten er zijn, en wat er door een bedrijf bij gebouwd moet worden. Dat bouwen doet de stichting dus niet, maar gebeurd door bedrijven. Vaak kan iets heel snel gebouwd worden, omdat er al heel erg veel is. Dat kan soms echt binnen vijf dagen.
De stichting GemGids werkt eigenlijk als volgt. Iedere overheidsorganisatie die mee doet moet betalen aan GemGids, dat is een soort van fee voor het gebruik ervan. Voor deze fee kunnen ze alle componenten gebruiken. Als ze door een bedrijf nieuwe componenten laten ontwikkelen worden deze ook van de stichting en kunnen andere overheden deze componenten ook weer gebruiken. De fee is 20 cent per inwoner. Een kleinere gemeente betaald minimaal 4500 EUR en een grote gemeente maximaal 9500 EUR jaarlijks. Met de kleine fee die betaald wordt zorgt de stichting voor beperkte ondersteuning, configuratie, hosting. De stichting is non-profit, we gebruiken het meeste voor ondersteuning, maar ook een deel voor communicatie en coördinatie. Maar ook de bedrijven die mee ontwikkelen aan de componenten betalen aan de stichting. Als bedrijf kun je dus niet zomaar gratis aan de broncode komen, daarvoor moet wel betaald

worden. We willen ook uiteindelijk dat bedrijven de volledige kosten gaan betalen, en niet meer de overheden. Dan is de functionaliteit gratis beschikbaar voor overheden, en betalen bedrijven de community.

Hoe vinden organisaties elkaar?
Bijvoorbeeld de gemeente Utrecht kwam naar ons toe. We hebben een website, we staan regelmatig in de pers, we houden workshops. GemGids is ook een soort kennisplatform. Er is binnen de stichting heel veel kennis aanwezig over ook andere projecten in Nederland.
Vooral vanuit de workshops komen er veel belangstellenden die later dan ook vragen hebben over de mogelijkheden. Ook zijn er bedrijven die erg geïnteresseerd zijn. Er zijn natuurlijk ook bedrijven die niks zien in open source, maar steeds meer bedrijven wel. En zo'n bedrijf kan dan partner worden.

Organisatiestructuur
Dat is nog beperkt uitgewerkt. We hebben de stichting vanuit het bedrijfsleven opgestart. In de stichting zitten momenteel als initiatiefnemers en investeerders een communicatiebureau, een internetbedrijf (Max.nl – die beheren het framework), en Max.nl/overheid waar ik zelf mede eigenaar ben. We zijn wel met andere stichtingen die ook met open source en communityvorming bezig zijn aan het kijken om te gaan fuseren, en we kijken ook in hoeverre we overheden in de stichting willen hebben.
We zijn het dus nog verder aan het uitwerken. Dat geldt ook voor het framework, we willen dat het liefste helemaal open source hebben, maar dan moeten er ook partijen zijn die het willen onderhouden, anders heeft dat weinig zin.

Rollen van de verschillende deelnemers
Dat is nog allemaal groeiende. Er is coördinatie tussen de verschillende deelnemers en partners, en al er een aanpassing nodig is aan het framework dan komt dat bij de stichting binnen. Als het een generieke aanpassing is dat wordt die doorgevoerd. Maar het gebeurd ook wel eens dat er zo'n specifieke aanpassingen zijn dat het niet de moeite is om dat in het framework op de nemen. Het wordt dan wel gemaakt door een partner en beschikbaar gesteld aan iedereen. Wat in het framework komt, en wat specifiek wordt, dat is iets dat nog moet groeien, dat kunnen we nu nog niet weten.

Belangrijkste ontwikkelaars
Op dit moment zijn er twee bedrijven die voor de verschillende deelnemers aan de stichting toepassingen hebben gemaakt op het framework: Max.nl/overheid en Reinforce. Maar dat zal zich wel gaan uitbreiden. Maar ook als overheidsorganisaties zelf een goede PHP ontwikkelaar hebben kan dat goed. Maar vaak is de kennis binnen de overheid vrij gering op dit vlak.

Speciale tools
We hebben een website voor de stichting. Daar zit al mail in. En misschien willen we binnenkort iets als MijnEgem, voor projectmanagement. Maar we hebben nu nog niet heel veel geld voor communicatie, en dat vinden we nu ook nog wat minder belangrijk. Webmarketing doel we wel veel.

Is er een community?
Er wel heel duidelijk de gedachte om veel te doem met de community. In de statuten staat dat we een kennisplatform met kennisdeling zijn, en daarnaast het ondersteuning van software verzorgen.

Neemt de community uiteindelijk de rol van bedrijven over?
Ik denk dat de community zich steeds meer op processen zal gaan richten. Je hebt de software, maar hoe implementeer je dat, of hoe pas je het aan aan de wetgeving. Je hebt het product, maar dat is nooit klaar, het is een dynamisch iets. Dat is ook een rol van overheid: data beschikbaar stellen zodat iedereen via de interface erbij kan. Dat kan dan via GemGids, of binnenkort misschien via mobiele telefoons. In dat aspect is GemGids misschien ook wel iets tijdelijks. Een bestemmingsplan vraag je niet bij de gemeente op, maar bij je makelaar als je een huis koopt, of een bouwvergunning bij een klusbedrijf. Deze bedrijven moeten dat zorgen dat hun systemen koppelen met die van overheden. De overheid gaat op den duur geen software meer bouwen, maar laat dat aan de markt over.

Hoe is ondersteuning geregeld?
Dat gaat nu nog via de stichting, maar steeds meer nemen overheden direct contact met hun IT bedrijf op. Op dit moment hebben al die overheden nog een contract met de stichting GemGids waarin de ondersteuning wordt geregeld. En als stichting hebben wij dat dan weer uitbesteed. Gemeentes kunnen bijvoorbeeld bellen als dingen het niet meer doen, als de website uit de lucht is, of als zaken door een nieuwe softwareversie niet meer werken.

Hoe groot kan het worden?
Geen idee, het kan heel groot worden, maar ook heel klein blijven. Aan de hand van de reacties die ik nu krijg denk ik dat het groot kan worden. Maar in overheidsland weet je het nooit. We praten nu bijvoorbeeld ook met GemNet, die heeft bij elke gemeente een beveiligde internetverbinding liggen voor berichtenverkeer. Dat is een goede partij, wij kunnen de diensten leveren, zij de beveiliging. Maar we praten ook met CMS leveranciers, en niet alleen van open source CMS systemen.

Is er interesse in open source, of alleen in GemGids al interessante oplossing?
Dit verschilt per deelnemer en partner. Wat we nu zien is dat het gebruik van open source software en vooral open standaarden wel een belangrijk beslispunt is bij partijen om zich aan te sluiten bij de GemGids community. Uiteraard blijven de innovatieve oplossingen van GemGids ook zeker de aandacht trekken.

Is er een speciale licentie?
Nee, op dit moment nog niet. We praten op dit moment wel met mensen van OSOSS wat voor soort licentie we zouden moeten nemen als we open source gaan. Bijvoorbeeld de GPL. Zelf zitten we meer te denken aan de LGPL omdat dat wat meer vrijheden geeft voor bedrijven om te investeren in GemGids.

Is het dan nog wel open source?
Nee, GemGids zelf niet, maar alle onderdelen – Linux, Apache, Mapserver, MySQL, PostgreSQL PHP – zijn wel allemaal open source. Maar dat willen we wel open source zijn, maar kan nu gewoon niet vanwege support. Maar voor alle deelnemers binnen de stichting is het open source, want zij kunnen alles gebruiken dat al een keer ontwikkeld is. Maar als alle bedrijven het gaan betalen dan kan het helemaal open source.

Is de stichting altijd nodig?
Ja, je zult altijd een coördinatiepunt moeten hebben, een soort autoriteit. En naast de stichting ook een audit-organisatie. Maar daar kun je nu wel veel geld insteken, maar we weten nog niet of het een succes wordt, dus dat bekijken we eerst. We denken al wel na over de toekomst, maar willen daar nog niet te veel in investeren. Dus laat het eerst maar slagen. We zijn gegroeid uit Pragmatika, en bekijken alles heel pragmatisch. Ook de overheadkosten die we nu hebben, die zijn heel laag. Bij andere projecten wordt er eerst veel geld in gestopt om een organisatie op te zetten. En het resultaat is bij GemGids hetzelfde, of soms beter.

Is het zo snel gegaan vanwege die pragmatische aanpak?
Ja, zeker.

Wat zijn de voor- en nadelen van open source?
Er zijn geen nadelen. We doen er veel mee, en dat gaat altijd goed. Beheer is wel altijd een issue. Bij veel projecten zit er maar een man achter, en dat is een gevaar. Juist daarom is GemGids nog niet helemaal open source, want dan kunnen we continuïteit beter waarborgen wat voor de deelnemers van levensbelang is.
Wat dat betreft is open source ook niet veel anders als andere software. Iedereen heeft behoefte aan bepaalde generieke software, en steeds meer wordt op die software specifieke dingen gemaakt. Maar daardoor worden wel steeds de communities kleiner. Iedereen heeft behoefte aan iets als MySQL, dus daar zijn ook veel ontwikkelaars voor te vinden. Maar die andere projecten hebben kleinere communities, en als de community kleiner is, is

de kans op slagen van het project ook kleiner. Maar ik denk dat uiteindelijk open source ook op dat kleinere niveau zal doorbreken.
Als voorbeeld, een tijd geleden kochten wij een licentie voor het tonen van kaartjes binnen de gemeente voor 70.000 gulden, en dat was alleen intern. Nu is die applicatie open source, omdat er zoveel concurrentie is dat ze het gewoon niet meer kunnen verkopen. Hetzelfde zie je nu voor loketten en CMS systemen.

Overheid verplicht open source te gebruiken?
Nee, denk ik niet. Voor de gemeente Voorst is een keer een applicatie ontwikkeld, en die applicatie is later verkocht aan het bedrijf dat het ontwikkeld had, die het weer bij een zestigtal andere gemeentes heeft geïnstalleerd. Maar daardoor had de gemeente wel voor altijd gratis support van dat bedrijf, dat hoorde bij de deal. En daardoor zijn de kosten voor de burger ook laag, en is er een goed product voor andere gemeenten, want het kost niet al te veel. En als het bedrijf wel te veel gaat rekenen zullen ze waarschijnlijk ingehaald worden door een open source systeem. Als je je beheer beter kunt regelen in een gesloten omgeving dan moet je dat doen. Ik vind wel dat de onderliggende systemen zo veel mogelijk open source moeten zijn.

Wat zijn je verwachtingen van het onderzoek?
Meer samenwerken en pragmatisch werken vanuit Den Haag. Er wordt heel veel gepolderd, eerst veel praten en dan pas actie ondernemen. Als je een fractie van dat geld gebruikt en dan tien acties uitzet, dan komt er vast wel eentje boven drijven in de markt.

F.6 MMProject

Interview Ted van Geest
13 juni 2007

Korte beschrijving van het project
MMProject is een doorontwikkeling van EGEMProject. Die hadden al projectsoftware waar je bestanden op kon zetten en delen. We gebruikte eerste een ander softwarepakket. Daar liep de licentie van af, en de kosten waren daarvoor vrij hoog. Die software werd vooral gebruikt om kennis te delen met gemeentes en ministeries. Toen zijn we eens gaan kijken of we dat niet konden vervangen. We keken ook naar open source omdat dat kostentechnisch voordelig was. EGEMProject had in basis de functionaliteit die we wilden, alleen niet uitgebreid genoeg. EGEMProject was al open source, en stond op de site van MMBase als apart project. We hebben dat door een van onze ontwikkelaars laten doorontwikkeling. Er is een betere mapstructuur bijgekomen, beter gebruikersbeheer, en authenticatie.
Het wordt niet heel intensief meer gebruikt, maar alleen door ons. We wilde het wel terugplaatsen in de open source community. We hebben hier ook OSOSS voor benaderd over hoe we dat het beste konden doen, maar daar is nooit iets uitgekomen. Daarom zijn de verbeteringen nooit gedeeld met anderen. Het was voor ons ook moeilijk te bepalen hoe we het terug moesten zetten. Het was in principe een doorontwikkeling, maar wel zo rigoureus veranderd. Dus is het dan een nieuwe versie of een heel nieuw project? Ook was er onduidelijkheid over de oorspronkelijke licentie.

Extern laten ontwikkelen
Ja, dat laten we altijd doen. We hebben het laten ontwikkelen door een bedrijf dat al ervaring had met MMBase.

Wat is de relatie tussen MMBase en MMProject?
MMBase is het framework waarin het gebouwd.

Tijd van ontwikkeling
Het heeft in totaal een paar maanden geduurd. Maar dat kwam ook omdat we binnen eOverheid goed in kaart wilde brengen wat we precies nodig hadden. De uiteindelijke ontwikkeling is iets van twee maanden geweest. We wilden ook niet heel rigoureuze dingen, maar meer allerlei verbeteringen.

Gebruikt door andere organisaties
Er zijn wel andere organisaties die interesse hebben getoond, en die wilde ook op de hoogte blijven van de ontwikkelingen. Maar ze hebben ons nooit benaderd dat ze ook daadwerkelijk de code willen hebben.

Is EGEMProject doorontwikkeld?
Niet dat ik weet. Dus volgens mij heeft er geen wederzijdse beïnvloeding plaatsgevonden. Ik weet eigenlijk ook niet of EGEM nog gebruikt maakt van EGEMProject.

Leiding over het project
We hebben met de ontwikkelaar een doorlopend contract. In het begin zaten er wel een aantal fouten in en wilde we nog wat extra functionaliteit. Dan kunnen we dat gewoon doorgeven aan de ontwikkelaar, en die voert het uit. Wij besluiten dus gewoon wat we nodig hebben. We hebben ook een tijdje over de documentatie lopen steggelen. Uiteindelijk is er een vrij summiere documentatie. De applicatie draait nu bij onze internetprovider. Maar de software doet nu precies wat ie moet doen, dus er hoeven geen veranderingen meer plaats te vinden.
Na de release hebben we ook bij alle gebruikers geïnformeerd wat er beter zou kunnen. Het was dan de bedoeling dat er een tweede release zou komen. Die is er nooit gekomen omdat er blijkbaar weinig behoefte aan was aan die extra functionaliteit.
Een aantal projecten gebruiken het nu nog wel. Maar er zijn bepaalde fases in het project waarin je documenten wilt delen. Daarna staan de documenten er nog wel, maar is het minder behoefte om veel te delen. Ook de projecten zelf hebben weinig behoefte aan doorontwikkeling, al zou daar wel een budget voor zijn, mochten ze veranderingen willen. Maar je ziet ook dat sommige projecten documenten gewoon via de mail delen.

Contact met MMBase, veranderingen in MMBase
Ik weet niet precies welke versie van MMBase wij draaien. We hebben het gewoon gedownload en geïnstalleerd. Maar we installeren niet steeds de nieuwe versie. Dat is ook niet nodig. De serviceprovider houdt het wel bij mochten er beveiligingsissues zijn.
We hebben ooit gekozen voor MMBase omdat andere websites van ons daar ook mee draaiden, en omdat EGEMProject daarop gemaakt was. Maar die andere websites houden nu op om MMBase te gebruiken, dus de betrokkenheid wordt wel steeds minder. Uiteindelijk had het dus net zo goed een PHP applicatie kunnen zijn. Dus als we die hadden kunnen krijgen, hadden we dat ook net zo goed kunnen doen.

Waarom wilde jullie het terug plaatsen in de community
EGEM had dat ook gedaan, en het doorontwikkelen en teruggeven aan een open source community sprak ons wel aan. Het is niet gebeurd, maar het kan nog steeds. Maar als iemand ons benaderd dan kunnen ze het gelijk krijgen. De open source gedachte vind ik zelf aantrekkelijk, en ook binnen ICTU wordt het veel gebruikt.

Bereid anderen actief te helpen
Als we het teruggeven dan is het als “as-is”. We hebben een beperkte documentatie, maar we hebben niet de resources om support te geven. De echte technische kennis ligt ook bij de ontwikkelaar. Zij kunnen dat wel support geven, maar daar moet dan wel voor betaald worden. Dat is misschien ook wel de reden dat ze minimale documentatie hebben gemaakt.

Organisaties die interesse kunnen hebben
Het is een heel brede tool, die voor veel organisaties interessant zou kunnen zijn. Zeker niet alleen de overheid. Maar aan de andere kant, er zijn heel veel soorten projectsoftware, vooral in PHP. Als een organisatie al MMBase hebben draaien kan dit wel een makkelijke tool zijn. Maar alles dat we hebben gemaakt is wel in het Nederlands, dus het zou alleen makkelijk binnen Nederland gebruikt kunnen worden.

Hoe is ondersteuning geregeld
Zodra de applicatie draait is de serviceprovider verantwoordelijk. Het systeem is heel stabiel en de fouten zijn er uit, dus de ontwikkelaar hebben we eigenlijk niet meer nodig.

Wat voor soort ondersteuning is behoefte aan
Eindelijk heel weinig, want het is een heel stabiel systeem. Ik doe intern de ondersteuning, zoals het aanmaken van projecten en rechten toewijzen. Als er nieuwe mensen komen leg ik ook uit hoe de software werkt. Dat kan meestal in een half uurtje.

Waarom open source gekozen
Vooral omdat het vanuit ICTU gestimuleerd werd om voor open source te kiezen. En EGEM zat dicht bij, dus we konden zo binnenlopen voor vragen.

Welke kennis wilde je van OSOSS krijgen voordat je het terug zou plaatsen?
Vooral de juiste contacten. We hadden wel een mailtje kunnen sturen, maar we dachten dat OSOSS de contacten met MMBase zou hebben. We verwachtte dat zijn de expertise zouden hebben over hoe je dat precies allemaal zou regelen. Dat kun je natuurlijk ook wel zelf uitvinden, maar je hebt ook nog je dagelijkse werkzaamheden. En OSOSS heeft volgens mij ook gezorgd dat EGEM hun project open source maakte, dus dat was voor ons de logische weg.
Er was bijvoorbeeld veel onduidelijk over de licentie. We wilde het echt terugzetten als “as-is” zodat we niet aansprakelijk zouden zijn.

Nadelen open source
Bij dit project niet. Meestal komt de open source community zelf met een antwoordt op de nadelen die er eerst zijn. Maar dan moet het project wel een redelijke omvang hebben. Wat dat betreft zou het voor anderen misschien wel een risico zijn MMProject te gaan gebruiken.

Genoeg stimulatie van open source
Je moet niet open source gebruiken om het open source. We gebruiken nu OpenOffice, en daar loop ik niet tegen problemen op.

Overheid anders dan bedrijfsleven
De overheid is belastinggeld, dus daarom heb je een iets andere verplichting. Je moet de beste deal maken, en als het goedkoper kan.

Verwachtingen van dit onderzoek
Lastig in te schatten. MMProject is niet zo heel groot. Ondersteuning kan vaak nog op ad hoc basis. We maken wel nog van wat andere kleine open source dingen, maar daar kunnen we gewoon terecht bij de site van de auteur. Het is natuurlijk dan wel een gevaar als diegene met de meeste kennis uit de organisatie verdwijnt. Ik weet nu veel van MMProject, maar andere niet. Je moet dus de documentatie heel goed vastleggen. Daar schort het bij MMProject ook nog wel aan.

F.7 Modernisering GBA

Interview Michel Bouten
4 juni 2007

Korte beschrijving van het project
De GBA is de gemeentelijke basis administratie. Daarin wordt door de gemeentes de persoonsgegevens beheerd van alle burgers in Nederland. Dat kent twee uitdagingen. Een, dat dat beter moet worden ondersteund bij de gemeentes zelf. En twee, dat de afnemers, zoals politie en belastingdienst beter bedient worden.

Wat hebben jullie precies gemaakt
Wij ontwikkelen de software die nodig is om de informatie te verwerken. En we maken de software om het centraal ter beschikking te stellen en om dat alles te beheren. Dat doen we met standaard tools, Java, de software die we maken is geheel open maar niet muteerbaar. Het is een zgn. white box, gemeentes mogen precies zien hoe het werkt, maar ze mogen er niet aankomen. De reden daarvoor is beheersbaarheid. Het moet bij alle ca. 450 gemeentes draaien, en als er per gemeente wijzigingen worden doorgevoerd is het niet meer beheersbaar.

Door wie wordt het gemaakt
Dat doen we in nauwe afstemming met gemeenten zelf. We ontwerpen, maken en testen, en BZK/BPR neemt het in beheer.

Waarom helemaal zelf
Omdat het bouwen eigenlijk maar een beperkte hoeveelheid capaciteit kost ten opzichte van het ontwerpen en testen. Het beschrijven van wat je moet doen is direct gerelateerd aan wat er in de wet staat. Dat beschrijven is vaak erg complex. Maar zodra we het hebben beschreven valt het bouwen ervan erg mee. Als het klaar is komt het bij een beheerorganisatie te liggen.

Maak je gebruik van standaard software
Uitgangspunt is dat als er standaard componenten zijn we die ook gebruiken. Onze ontwikkelomgeving bestaat uit standaard componenten. Dat is vaak niet open source omdat het ingewikkelde pakketten zijn. Maar als er open source is wat voldoet aan onze eisen gebruiken we dat ook. We volgens dus het kabinetsbeleid daarin. We maken gebruik van een open source database, PostgreSQL. We programmeren in Java, omdat we het zo open mogelijk willen doen.
We maken maatwerk, omdat het direct uit de wet te herleiden is wat wij moeten maken. Daar is dus ook geen standaard software voor.

En de software die gemeentes nu gebruiken?
Dat is commerciële maatwerk software. Er zijn momenteel een drietal leveranciers. Hun business case is dat ze het een keer maken en dan kunnen verkopen aan 100 tot 200 gemeentes. Als onze software er uiteindelijk is zijn gemeentes verplicht die te gaan gebruiken. Wij maken een deel van de software, de database en de services (de kern van het Burgerzakensysteem). Maar de interface om die services aan te roepen en de integratie met de overige processen binnen de gemeente, die maken marktpartijen. En er zijn nu al leveranciers die de interface software willen gaan maken. En dat willen ze in ieder geval ook in Java gaan maken, omdat het dan beter past.

Leiding over het project
De opdrachtgever regelt het bestuurlijke. En wij hebben het inhoudelijk en technisch in de hand. We stemmen dat wel af met de partij die het in beheer moet gaan nemen, en met de omgeving.

Organisatiestructuur
We hebben een werkgroep van zeven gemeentelijke vertegenwoordigers. Die kennen het hele proces bij de gemeente. We hebben alle 116 processen in beeld. En die vertegenwoordigers komen hier twee dagen per week om alles te bespreken. Die interactie is dus best wel intensief. We zitten nu in het proces om de eerste stappen te maken. We hebben de database ingericht, en de omgeving, en nu moeten we de software gaan maken. Die moet dan opgeleverd worden zodat de leveranciers de interface kunnen bouwen, daarom moet het ook open zijn.

Planning vooraf/ad hoc
Er is van tevoren afgesproken welke functionaliteit er in komt. Dat heet bij ons de definitiestudie. Maar dat is zo'n abstractieniveau dat er nog veel wijzigingen voor de specifieke realisatie uit kunnen komen. We hebben GBA processen, maar ook niet-GBA processen. Een paspoort verlengen is bijvoorbeeld een niet-GBA proces. Maar dit heeft wel gegevens uit de GBA nodig. Dit soort processen wijzigen soms nog wel.
De leveranciers hebben geen zeggenschap over de functionaliteit. De gemeentes moeten aangeven welke functionaliteit ze precies willen hebben. Die gemeentes worden daar overigens wel bij geholpen door de leveranciers. Maar als alle leveranciers zeggen “doe het zo, dan is het makkelijker”, dan doen we dan.
Maar door veranderende wetgeving moeten soms ook dingen worden aangepast. In de huidige software was het doorvoeren van een wijziging (bijv. homohuwelijk) complex en langdurig, omdat man-man of vrouw-vrouw in het oorspronkelijke gegevensmodel voor het huwelijk niet aan elkaar gekoppeld kon worden.

Financiële zaken
Er is een bestuursakkoord gesloten tussen het rijk en de gemeentes.

Community
De community is meer voor de componenten die je gebruikt om de maatwerksoftware te maken. Ik heb geen community nodig, want zo wordt deze software niet gebouwd. Want eigenlijk heb je het over open standaard software die je met communities bouwt. Maar als een deel van onze software beter kan met zaken die door een community worden gemaakt, dan nemen we dat mee. Wij dragen ook actief bij aan de software die we gebruiken. Wij hebben laatst fouten uit PostgreSQL gehaald. In zoverre maken wij dus wel deel uit van de community van andere projecten.

Code van andere accepteren
Als iemand aangeeft dat de code anders en beter kan zou je gek zijn als je als overheid daar niet op in zou gaan. Dat is wat anders dan dat het op 450 plaatsen vrijgeeft om te laten muteren. Want dat krijg je net zoiets als alle versies van open source systemen die er zijn, en dat is niet meer te beheren.

Hoe vinden organisaties elkaar
De NVVB en de VNG trekken het samen, en er hebben zich al zes leveranciers gemeld die het Burgerzakensysteem willen ontwikkelen. Wij maken dus de documentatie en de services voor de leveranciers. En de gemeentes hebben nu onder leiding van de VNG en de NVVB zelf een project gestart om de processen binnen de gemeentes zo standaard mogelijk te maken. En dat is weer input voor de leveranciers. De gemeenten hebben voor een groot deel dezelfde processen en informatie nodig. Dus ca. 90% van alles is standaard. Afwijkingen zijn bijv. die gemeenten die te maken hebben met het Koninklijk Huis, waarvoor processen specifiek worden ingericht.
De interface bouwen is ook nog erg lastig. Als we het centraal gaan maken dan kunnen we niet aan alle variëteiten voldoen. Het echte werk zit 'm er ook in om allerlei andere systemen bij de gemeentes te integreren met dit systeem.

Ondersteuning
Er zijn gebruikersverenigingen voor de systemen die er nu zijn. De leverancier zal ondersteuning leveren voor zijn deel, maar zal moeten terugvallen op het centrale gedeelte als daar problemen zijn. Als er centraal wijzigingen nodig zijn dan is er een overlegcircuit om dat mogelijk te maken. In de huidige systemen duurt dat 3 tot 12 maanden, en in de nieuwe situatie moet dat in ieder geval een stuk sneller gaan.

Waarom open source
We willen de software zo laagdrempelig mogelijk bij de gemeenten naar binnen brengen. Als we het dan niet met open source maken moeten ze licentiekosten gaan betalen. We maken momenteel gebruik van Java omdat het robuust genoeg is om op te ontwikkelen. Er zijn maar twee mogelijkheden: .NET en Java, en dan kiezen we conform het kabinetsbeleid voor de open software.

Hoe reageren gemeentes op open source
Over het algemeen positief. We hebben een proof of concept gemaakt waarin we laten zien dat het werkt. Het gaat er om dat we het werkend kunnen krijgen en kunnen houden. En er is genoeg kennis over Java in Nederland om dat te garanderen. Dan maakt het gemeentes niet echt meer uit in welke taal het gemaakt is.

Voor- en nadelen van open source
Het enige nadeel is dat het instrumentarium dat we gebruiken nog niet helemaal open source is.
Ik heb vroeger ook aan projecten gewerkt die niet open waren. Dan kwam er een nieuwe release die je niet in de hand had, of de vorige release werd niet meer ondersteund. Dan wordt je dus gedwongen om over te schakelen. Dit kun je doorbreken door een lijn van één leverancier te kiezen. Dat gaat echter ook niet zonder slag of stoot en is voor overheden met aanbestedingsplicht lastig te realiseren. Maar door voor open source te kiezen houd je het helemaal in eigen hand. Ik zie dus heel veel voordelen voor open source software, zeker bij maatwerksoftware.

Overheid en open source
Ja, al is het alleen maar om minder afhankelijk te zijn van partijen. Overigens zijn open standaarden voor de interoperabiliteit veel belangrijker dan open source software. Maar open standaarden gebruik je automatisch bij open source software. En dat hoeft niet altijd zo te zijn bij closed source software.
Ook bij het aanbesteden is het van belang. Als overheid moeten we eens in de zoveel tijd opnieuw aanbesteden. Als je dan vast zit aan een leverancier moet je hoge kosten maken als je een andere leverancier krijgt. Commerciële partijen kunnen een partnership aangaan er voor kiezen om bij een partij te blijven, dus die hebben daar minder last van.
Waar ik denk dat wel nog een rol ligt voor de overheid is in het actief stimuleren van de community van een aantal projecten. Met name die producten die we allemaal gebruiken, zoals voor een database, een financieel pakket en een office pakket. We kiezen nu vaak geen open source pakket omdat het net niet kan wat willen, maar willen er niks bij bouwen. En we willen graag naar een leverancier kunnen stappen als het niet werkt.

Verwachtingen van het onderzoek
Voor alle potentiële gebruikers van software, ook voor projecten die maatwerksoftware maken, is het goed te weten welke software beschikbaar is in open source. Dus moeten de ontwikkelingen van van open source goed in kaart worden gebracht.

F.8 Op Afspraak

Interview Carien van Leeuwen
4 juni 2007

Korte omschrijving van het project
Met Op Afspraak is het de bedoeling dat de burger online een afspraak kan maken met de gemeente. Zoiets bestaat al wel, maar wij willen dat het een generieke module wordt, leveranciersonafhankelijk en open source. Op Afspraak wordt een soort stekkerdoos die we bij gemeentes aansluiten op agenda-systemen. Naar de kant van de burger moet er een koppeling worden gemaakt met naam en adres gegevens en moeten er producten worden getoond. Producten die ondersteund gaan worden zijn paspoorten, identiteitsbewijzen, rijbewijzen, de begrafenisagenda en een agenda voor de WMO.
De gemeente Den Haag is nogal leading in open source en wil het open source hebben. Daar is het idee ook ontstaan, en ze wilde dat graag met andere gemeentes oppakken.
Atos Origin is het bedrijf dat de opdracht uiteindelijk is gegund. Om de kosten de drukken zijn zij een alliantie aangegaan met eMax, die al een vergelijkbaar product had. Dat bleek uiteindelijk nog wel tegen te vallen in hoeverre dat vergelijkbaar was. Maar nu zitten we in een hele spannende fase: gaat het echt werken zoals we willen, en is het wel genoeg open source.
Het is uiteindelijk de bedoeling dat de hosting van het hele product centraal gaat gebeuren, zodat beheer erg makkelijk is.

Waarom extern ontwikkeld?
GovUnited bestaat alleen uit projectmanagers, we laten alles extern uitvoeren.

Waarop open source?
GovUnited staat voor open source, dus alles wat we laten ontwikkelen wordt open source. Dan kunnen ook kleine gemeentes profiteren van kennis van grote gemeentes. Ons hele business model is open source, generieke software, zoveel mogelijk gezamenlijk optrekken, en delen van kennis. Het is ook een beetje de filosofie of magie van open source. Alle partijen reageren ook erg positief op het feit dat het open source is. Maar er zijn ook een aantal gemeentes die juist daarom niet mee doen. Andere gemeentes zeggen weer dat open source wel goed is, maar ASP niet, waarin Op Afspraak geschreven wordt. Hun architectuur is daar niet op ingericht.

Hoe vinden partijen elkaar?
Gemeentes moeten lid zijn van GovUnited. Alle leden worden op de hoogte gehouden van alle projecten, en worden uitgenodigd om mee te doen aan projecten. Een project is voor ons pas levensvatbaar als er genoeg deelnemende gemeentes zijn. Voor Op Afspraak zijn er nu vijf gemeentes, en dat is denk ik ook een goed aantal om mee te starten. Anders wordt het al snel onoverzichtelijk. Uiteindelijk moeten er natuurlijk meer gemeentes bij komen.
Als GovUnited bestaan we sinds januari. We hebben bijvoorbeeld ook het aanbestedingstraject Andez3. Hierover houden we veel bijeenkomsten, en hierdoor komen er steeds meer leden bij.

Leiding over het project
In de opstartfase implementeren we het bij vijf gemeentes. Als dat succesvol is zal er een nieuwe offerte uitgaan voor het beheer van de software, dat hoeft niet per definitie Atos Origin te zijn. Dit omdat we het leveranciersonafhankelijk willen hebben.
Voorafgaande aan het project is er een haalbaarheidsonderzoek gedaan door een bedrijf. Die hebben onderzocht hoe we het beste zoiets konden aanpakken. Dat heeft ook als basis voor de offerte gediend. Daaruit komt ook ons hele model met deliverables voor.
Verder hebben we een projectgroep waarin alle deelnemers kunnen meepraten. We maken gebruik van de Agile methodiek. Daarvoor heb je een gebruikersgroep nodig. Die hebben veel gepraat over de interface. En elke maand is die groep bij elkaar geweest, is er feedback uitgekomen, en dat is naar Atos Origin gestuurd. Uiteindelijk heeft de Agile methodiek gediend om het hele project helder te krijgen. Maar dan moet je stoppen met de methodiek, anders blijven we hangen in iteraties en komt er nooit een product.
De structuur van het project is nu heel open. Alle deelnemers zitten met Atos Origen om de tafel. Ik sta wel op het punt om dat meer te gaan scheiden. Soms is het namelijk in het belang van Atos Origen om iets niet te gaan bouwen. Als ze dan voelen dat er bij iemand van de deelnemers twijfel is over wat er gebouwd moet worden gaan ze dat natuurlijk helemaal uitspelen. Ik wil dus eerst overleggen met de deelnemers, en dan gaan wij daarna wel overleggen met de leverancier, want om de een of andere reden willen ze je toch altijd uitbuiten.

Financiële zaken
De gemeentes betalen allemaal mee aan de ontwikkeling, en GovUnited heeft zelf ook nog wat geïnvesteerd in het project. En als nieuwe gemeentes zich aansluiten bij het project zal er zeker een en ander verdisconteerd worden. Er zullen altijd handelingen gedaan moeten worden om het aan te sluiten op die gemeente, dus daar moeten ze dan voor betalen. Maar die bedragen liggen natuurlijk vele malen lager dan dat elke gemeente het zelf moet gaan doen. Het hele project kost nu 160.000 euro, en dat kan een kleine gemeente nooit betalen.

Feature requests
We hebben nog niet echt goed nagedacht over wat er in de toekomst gaat gebeuren als een gemeente extra functionaliteit wil. Ik weet niet of een gemeente dan naar een willekeurige partij kan stappen. Waarschijnlijk komt dat helemaal bij GovUnited te liggen. Als gemeentes iets willen dienen ze dat in, kijken we of er genoeg draagvlak voor is en dan laten we het uitvoeren. Als er dan meerdere gemeentes belangstelling voor hebben kunnen de kosten ook gedeeld worden. Ander probleem is wel, als we met steeds meer gemeentes aan tafel zitten, dat het lastiger is om tot besluitvorming te komen.

Toekomstvisie
We beginnen met vijf gemeentes. Maar dat is natuurlijk maar het begin. Ik denk dat wij wel altijd de leiding over het project zullen blijven houden, natuurlijk wel in samenspraak met de gemeentes.

Ondersteuning, ruimte voor meerdere bedrijven
Dat kan ook bij meerdere bedrijven, maar wij besteden het bij een bedrijf aan. Maar als de deelnemers het bij een andere partij willen beleggen kan dat. Het is voor veel bedrijven interessant om voor gemeentes te werken, en zeker als het open source is. Dus ik zie niet in waarom andere bedrijven het niet zouden willen.
Dan is er natuurlijk wel meer coördinatie nodig. Daar moeten we dus van te voren over nadenken en regelen hoe we dat willen.

Voor- en nadelen open source
Ik denk dat die te vergelijken zijn met closed source. Ik vind vooral de filosofie erachter goed. De rest weegt wel tegen elkaar op.

Open source en overheid
Ik denk dat het voor iedereen geldt. Maar het is wel heel goed dat de overheid hier het voortouw in neemt.

Buitenlandse interesse?
Mogelijk is het ook interessant voor buitenlandse organisaties, maar alles is nu in het Nederlands. GovUnited is erg gericht op Nederland, dus ik denk niet dat we heel snel naar het buitenland gaan.

Verwachtingen ten aanzien van het onderzoek
Meer over technisch beheer. Is er iets van een checklist zodat je je kunt voorbereiden.

F.9 Watlas

Interview Piet Feddema
25 april 2007

Korte beschrijving van het project
Interwad is een ICTU programma sinds 2002, maar bestaat al sinds 1997. Het doel is om Waddenzee-informatie en kennis te ontsluiten via het internet. Dat doen we in opdracht van Waddenoverheden. Onze opdrachtgevers zijn vijf ministeries, drie provincies, en 18 gemeentes. Ze betalen ook het programma. We werken hier met ongeveer 3 FTE. Ook hebben we als doel communicatie tussen de verschillende overheden die onze opdrachtgever zijn. Maar het eerste doel is tegenwoordig belangrijker.

Wat is het ontwikkeld
Voor de informatievoorziening hebben we de Watlas. Hierop laten we digitale kaarten zien. Dit begint langzamerhand uit te groeien tot een echte mapserver. Hierop is informatie van verschillende overheden te zien, en deze informatie kan ook weer door verschillende overheden gebruikt worden.
We hebben ons altijd geconcentreerd op internet. Het is de bedoeling dat iedereen vanuit zijn doelgroep bij de juiste informatie kan komen. Onze doelgroepen zijn het grote publiek, onderzoekers, ambtenaren, en bestuurders.
Het kaartmateriaal is een manier van informatie ontsluiten. Een kaart is een makkelijke manier om van daar uit weer bij geschreven teksten en foto's uit te komen. Dat idee hebben we vormgegeven in de watlas. Hier zijn we in 1998 al begonnen met ArchIMS. Hier zat wel een licentie aan vast. Ons content management systeem had ook een licentie. Later werden we getriggerd door open source, vanwege het ontbreken van licentiekosten en het hebben van wereldwijde communities. En als het dezelfde functionaliteit heeft maar minder kosten dan is de keuze makkelijk. Je moet dan vaak wel weer een externe partij inhuren om de configuratie te doen, om het in te passen in de organisatie. We maken voor de kaart nu gebruik van MapServer. Ook ons CMS is open source, namelijk Typo3. Deze twee hebben we ook aan elkaar gekoppeld, zodat de informatie goed doorzoekbaar is. Deze koppeling is extern voor ons ontwikkeld. Dat ontwikkelen ging vrij snel, ongeveer 10 mandagen werk.

Wordt het door meerdere partijen gebruikt?
Nee, niet dat ik weet. Maar die is wel beschikbaar, maar alleen voor andere overheden. Het hele systeem kan wat mij betreft één op één overgezet worden naar een andere overheid, mocht daar belangstelling voor zijn. Dat is ook voor ons de definitie van open source. We hebben alleen niet alles gestandaardiseerd, met open standaarden en zo. Daar hebben we op dit moment het geld en de mensen niet voor.

Hoe ga je om met updates van bestaande producten?
Vooral Typo3 heeft regelmatig nieuwe versies. Hier gaan we ook in mee, en hier investeren we ook in. Jaarlijks is dat zo'n 7000 EUR. Dat is vooral aan manuren om de conversie helemaal vlekkeloos te laten doen. Dit gebeurt ook altijd extern.

Omgaan met gedeelde software
Er wordt nu aangegeven dat alles van ons gebruikt kan worden. Als een organisatie zich meldt dan zullen we die helpen. Ook als ze een adviseur willen om het technisch te realiseren, dan kunnen we die inhuren. Die extra kosten berekenen we dan natuurlijk wel door. We hopen eigenlijk ook dat dat gaat gebeuren, want dat betekend dat wij ook gebruik kunnen maken van hun ervaringen.
Kaartmateriaal stellen we al beschikbaar in het GeoNetwork. Ook met andere overheidsorganisaties wisselen wij kaarten uit. We krijgen kaarten, of krijgen kaarten van hun. Rijkswaterstaat koopt diverse kaarten centraal in, en wij kunnen vragen of we die kunnen gebruiken, en dat kan altijd.
We hebben ook speciale monitoring applicaties laten ontwikkelen. Hiermee kunnen we bijvoorbeeld zien hoeveel zeehonden er in een bepaald gebied van de Waddenzee hebben gezwommen. Ook die tool is open source.

Actief achteraan gaan
We maken wel reclame. We hebben een adressenbestand van ongeveer 700 organisaties. Daar krijgen we informatie van en wisselen ook informatie mee uit. We hebben bijvoorbeeld een mailing gedaan. En als er reacties komen dan delen we onze know-how. Maar verder doen we niet heel veel daaraan. Onze core business is ook niet het verkopen van techniek.
En al onze metagegevens staan ook op een platform. Dus we hopen dat anderen dit oppakken en ook weer gaan verbeteren.

Soort organisaties die belangstelling kunnen hebben
De open source techniek die er achter zit is breed inzetbaar. Overheidsorganisaties die nu nog gebruik maken van Esri producten kunnen veel baat hebben bij onze software. Daar zit een licentie aan, en die beginnen bij 20.000 EUR.
In Duitsland werken ze nu heel veel met ArcIMS. We hebben wel internationaal aangegeven dat we goede technologie hebben. Maar een organisatie gaat niet zo snel om, dat is heel lastig. Ook de NAM heeft een uitgebreid GIS, maar ook met licenties, dus ook daar liggen mogelijkheden. Maar of ze willen omschakelen, dat weet ik niet.

Is er een community
Voor met name de mapserver, het uitwisselen van kaarten, met de goede metagegevens. Maar we hebben verder nog geen forum of mailinglist of zo. We draaien wel mee in een bestaande geonetwork community. En we moeten kijken wat de reacties daar van zijn. Als dat goed gaat gaan we ook andere overheden vragen hun kaartinformatie daar beschikbaar te stellen. En we leveren nu ook al kaarten aan onderzoeksinstituten en universiteiten. Vooralsnog is het zo dat we alleen aan een commercieel bureau leveren als ze een onderzoek doen in opdracht van een overheid. Er zit dan een contract bij de kaarten dat ze alleen voor dat doel te gebruiken zijn. Maar in de toekomst zou ik graag zien dat al het kaartmateriaal van de overheid voor iedereen beschikbaar is. Dat is nu nog niet zo omdat kaartmateriaal veel geld waar is.

Hoe is ondersteuning geregeld
Veel is uitbesteed. Maar we houden ook veel websites in de gaten. Bijvoorbeeld een website die veel CMS'en vergelijkt. Hier hebben we veel aan gehad. Verder hebben we eigenlijk alleen ondersteuning nodig als we wat nieuws willen. Maar dat hangt ook heel erg van het budget af dat we kunnen vrij maken. Maar als je eenmaal een keuze hebt gemaakt voor open source, zijn de problemen niet anders dan met andere software. In het begin was het nog wel even lastig om een bureau te vinden dat open source wilde ondersteunen, maar tegenwoordig is dat heel makkelijk. Ook in de toekomst verwacht ik weinig problemen omdat we twee producten hebben gekozen die volop in ontwikkeling zijn. Dan ben je altijd verzekerd van een community, en dus nieuwe versies. Als dit een grote organisatie was met genoeg budget dat zou ik ook wel genoeg ondersteuning kunnen krijgen van de community, zonder dat er een extern bureau nodig is. Dan gaat gewoon iemand van de organisatie meedraaien in de community. Dat is uiteindelijk efficiënter dan elke keer een adviseur inhuren.

Nadelen van open source
Bij MapServer helemaal geen. Bij Typo3, sommige dingen zijn soms een beetje onhandig. Maar je schakelt niet zomaar over. Maar dat zijn verschillen in functionaliteit en beheer, en niet zozeer algemene nadelen van open source.

Stimulatie om open source te gebruiken
Daar hoor ik wisselende dingen over, dat weet ik niet zo goed. Er is misschien angst voor de ongrijpbaarheid van een community. Maar ik hoor wel steeds meer goede geluiden.

Verwachtingen ten aanzien van het onderzoek
Een life cycle overzicht. En dat met een TCO, waar alles in wordt meegenomen. En dat afzetten tegen commerciële software. Wij zijn nogal een innovatieve club met goede adviseurs, maar een hele goede analyse hebben we nooit kunnen doen. Het was zeker voor 50% een gok om met open source aan de slag te gaan.

F.10 Interview Ruben van Wendel de Joode

15 januari 2007

Daniel Wat is de definitie van een actieve community, wat zijn de kenmerken daarvan?
Ruben Veel mensen. Als je het hebt over inzet, dan uiteraard via het internet. Als je het hebt over actieve communities is het belangrijk dat er ook daadwerkelijk iets geproduceerd word, dan kan zijn in het oplossen van vragen, of in het schrijven van software. Veel tools om het te ondersteunen. En wat ik zelf vooral mooi vind aan open source is dat gebruikers en professionals door elkaar heen lopen, er is een veel minder duidelijk onderscheid dan bij een bedrijf, waar je een bedrijf-klant-relatie hebt. Bij open source is er juist een samenwerkingsrelatie en staan gebruiker en professional juist naast elkaar. Een community heeft vaak ook iets waar rond ze georganiseerd is, bijvoorbeeld bepaalde thema of software die ze bind. Dit heeft vaak minder te maken met een gebruiksgebied maar eerder een bepaalde visie van de ontwikkelaars. Er zijn bijvoorbeeld geen communities rondom "overheid", maar rondom "CMS 1", "CMS 2" etc. Bijna alle succesvolle communities zijn inhoudelijk gedreven.

Daniel Welke zaken zijn echt essentieel voor een community?
Ruben Het is een beetje een kip-ei-verhaal. Een succesvolle community is een community die veel veel mensen heeft. Dus om het succesvol te laten zijn heb je veel mensen nodig, maar je kunt alleen veel mensen trekken door veel inhoud te hebben. En dat kun je alleen maar krijgen door veel mensen te hebben. Zulke communities ontstaan vaak vanzelf. Soms verbaast het je dat de ene community wel succesvol wordt en de andere niet. Je kunt wel heel veel doen in de support van een community, een bepaald aantal basisdingen moet goed zijn. Zoals een plek om je software op te slaan, mailinglijsten, bug report systeem voor een community van een bepaalde grootte. Als je kijkt naar de overheid, en je wilt iets intern gaan doen moet je zorgen dat er potentieel in ieder geval genoeg mensen zijn die er iets mee kunnen. In een heel obscuur, klein gebied hoef je geen community op te zetten, dat heeft vrij weinig nut. Wat ook heel belangrijk is, is dat een community linkt aan de direct werkzaamheden van mensen. Binnen de overheid kun je niks opzetten als mensen daarvoor heel ver buiten hun dagelijkse werkzaamheden moeten gaan denken.

Daniel Kun je overal een community voor opzetten, of zijn er bepaalde project waarvoor het geen zin heeft, bijvoorbeeld omdat ze te klein zijn? Of zijn twee mensen ook al een community?
Ruben Twee mensen kunnen al een community vormen, maar de community is dan heel erg kwetsbaar. Maar de vraag is ook: wat wil je er mee? Als je een community opzet dan is er meestal

de doelstelling dat je kennis wilt verspreiden onder andere mensen.
Als een community uit heel weinig mensen bestaat heeft het weinig zin om dat actief te gaan stimuleren. Als twee mensen alleen een community vormen dan zullen ze sowieso wel met elkaar gaan praten. Je moet alleen een community opzetten als het een bepaalde relevantie of draagvlak heeft. Een community voor de overheid moet ook iets hebben dat strategisch van belang is voor de overheid. Waarschijnlijk kost het opzetten van communities bij de overheid veel tijd, dus moet je het alleen doen als het ook echt relevant is. Maar er zijn natuurlijk ook communities die met twee beginnen en daarna heel erg groot groeien. In die zin moet je kleine communities gewoon hun gang laten gaan. Dat is bijvoorbeeld het mooie van een site als SourceForge, daar kan iedereen aan meedoen. Dat is ook precies open source, je bepaald zelf wat je belangrijk vindt. Er zijn wel projecten die eisen dat er een minimum aantal ontwikkelaars is dat aan een project begint. Apache doet dat bijvoorbeeld. Zij hebben een groot aantal projecten, en als je een nieuw project wilt beginnen heb je minimaal vijf (of zo) mensen nodig die een handtekening zetten dat ze het een goed project vinden en mee zullen helpen met de ontwikkeling. Je zou dat binnen de overheid ook kunnen doen, maar je kunt je dan ook direct weer afvragen of dat altijd handig is. Als er twee mensen enthousiast aan de gang gaan, en twintig vinden het resultaat geweldig, dan is dat misschien ook al voldoende. Je laat die twee dan eerst een tijd aanmodderen, en als ze na een tijd nog niks hebben dan geef je als overheid dat project geen prominente plaats meer.

Daniel Hoe kunnen partijen elkaar het beste vinden? Er zijn natuurlijk sites als SourceForge waar je zelf opzoek kunt naar interessante projecten, maar hoe zou dat binnen organisaties kunnen fungeren? Zullen organisaties ook actief gebruik gaan maken van SourceForge om projecten te zoeken?
Ruben Nee, voor bijvoorbeeld de overheid is het zaak om een eigen, interne site te hebben. Mensen moeten namelijk eerst opgeleid worden. Werknemers moet wennen aan het idee dat ze iets maken dat ze direct delen met anderen. Als je dan een systeem hebt waarbij mensen van over de hele wereld direct invloed kunnen uitoefenen op een product dan komt het waarschijnlijk niet van de grond. Daarnaast is SourceForge vooral heel handig voor mensen die veel ervaring hebben met ontwikkeling en communities, terwijl de meeste mensen binnen de overheid daar bijna geen ervaring mee hebben. Daarom moet een systeem voor de overheid toegankelijker maken en laten lijken op systemen die ze nu ook al gebruiken. Dus eerst intern, daarna uitbreiden naar extern.
Daniel Er is bijvoorbeeld al wel het uitwisselplatform, dat een speciale site is voor de overheid, gebaseerd op de SourceForge software.
Ruben En hoe werkt dat op dit moment?
Daniel Er worden een aantal projecten op ontwikkeld. Ik heb voor mijn case studies gekeken naar hoeveel ontwikkeling er nou daadwerkelijk plaatsvindt, en bij de meeste projecten is dat vrij weinig. De meeste forums en mailinglijsten zijn leeg.

Daniel Voor mijn onderzoek wil ik kijken naar de ondersteuning op open source software. Ondersteuning zij ik dan zowel als het helpen van andere gebruikers als het doorontwikkelen van de applicatie. Zorgt een actieve community automatisch voor een goede ondersteuning?
Ruben Geen enkele community garandeert iets. De meeste communities werken nog steeds op basis van vrijwilligheid. Als je zo'n community ziet moet je maar geluk hebben dat je geholpen wordt. Een naam hebben is natuurlijk wel handig. Als je dingen bijdraagt, en mensen kennen je daardoor, zullen ze je op andere punten ook hulp geven. Het is een economie die is gebaseerd op andere fundamenten dan geld, maar meer op elkaar willen helpen. Dus een dienst vraagt om een wederdienst, een soort wederkerigheid. Dus als je alleen maar iets neemt, en nooit iets geeft ("lurker") dan zul je op een gegeven moment niet meer geholpen worden. Maar in een actieve community zul je wel wat sneller geholpen worden, al wat het alleen maar zodat deze mensen kunnen laten zien hoe goed ze zijn. Wat grote communities wel goed doen, is dat ze, omdat ze ze groot zijn, en andere partijen omheen komen die die ondersteuning gaan verzorgen. Er ontstaan ook steeds meer Marktplaats-achtige systemen waarbij bedrijven beloning geven aan ontwikkelaars voor het maken van bepaalde zaken. Als je het hebt over de overheid, dan zullen mensen elkaar waarschijnlijk veel sneller gaan helpen, omdat je elkaar kent. Wat je ook heel vaak hebt is dat mensen de community gebruiken om werk te krijgen. Door hu werk in de community kunnen ze zich profileren, en laten zien wat ze kunnen. Als commerciële partijen dan op zoek zijn naar mensen met kennis van die software dan is het duidelijk welke mensen er het meeste van af weten.

Daniel Communities draaien op dit moment nog heel erg op vrijwilligers, of dat is in ieder geval het idee. Wat zal er veranderen binnen een communities als zich steeds meer commerciële partijen erin gaan mengen?
Ruben Het is een beetje als in een glazen bol kijken; niemand weet het. Er zijn drie scenario's mogelijk: er veranderd niks, of er veranderd heel veel, in negatieve of in positieve zin. Een community heeft verschillende outputs: software, hulp kennisdeling. Er zijn twee verschillende inputs: dan kan vrijwillig zijn, of commercieel. Een commerciële partij kan uiteindelijk niet veel veranderen aan de output, want dat moet via de community. De vraag is natuurlijk wat er gaat gebeuren als 90 procent van de ontwikkelaars bestaat uit commerciële bedrijven, en nog maar 10 procent uit vrijwilligers. Zolang er geen geld wordt betaald voor de output van de community zal er niks veranderen; je weet namelijk niet direct of je hulp krijgt van een commerciële partij of van een vrijwilliger. Als ontvangende bedrijven willen betalen voor bepaalde zaken, dan maakt het ook niks uit of die opdracht wordt aangenomen door een bedrijf of door een individuele programmeur.
Wat wel kan gebeuren is dat veel bedrijven zaken niet meer via de community gaan regelen, maar direct naar de commerciële bedrijven toestappen. Maar voor veel bedrijven is de community een hele goede tool om naamsbekendheid te krijgen. Dus blijven deze bedrijven bijdragen aan de community en verandert er niet veel.
Veel van de mensen die vrijwillig werken aan projecten doen dat op freelance basis, maar ook omdat ze het leuk vinden. Als ze dan het aanbod krijgen om er betaald aan te werken zullen ze dit doen, ze kunnen dan van hun hobby hun werk maken. Kijk bijvoorbeeld naar de stichting van Apache, daarin zitten bijna alleen maar mensen die in dienst zijn bij grote bedrijven.
Daniel Maar blijven communities voor grote projecten dan wel bestaansrecht hebben, als de commerciële ontwikkelaars eenmaal hun naamsbekendheid hebben?
Ruben Partijen werken nu binnen een community samen, waar ze vroeger misschien alleen tegen elkaar hadden geconcurreerd. En zolang partijen zich realiseren dat ze elkaar nodig hebben zullen ze actief blijven in de community. Een mooi voorbeeld is MMBase, daar zit Finalist in, en ze zijn duidelijk de grootste partij. En zij blijven actief in de community, om te laten zien dat ze "open source" zijn, dus als marketingkanaal. Daarnaast zijn ze de beste, en kunnen ze dus dingen prijs geven, omdat andere er vrij weinig mee kunnen. Als klanten kwaliteit willen zullen ze dus naar Finalist toe stappen. Ze kunnen ook heel veel prijsgeven, omdat het een CMS is, en het meeste werk zit er in om dat helemaal op maat te maken voor de klant.
Daniel Gelden aan aantal reden die je in je boek noemt, zoals prestige voor ontwikkelaars wel nog als ze voor bedrijven werken, of geldt dat dan alleen nog maar op bedrijfsniveau? Het zijn noch vooral individuele redenen.
Ruben Bonaccorsi & Rossi, en Wouter Stam van de VU hebben onderzoek gedaan waarom bedrijven meedoen aan open source.
Er zijn eingelijk twee soorten bedrijven. Ten eerste zijn er de bedrijven die tegen haar werknemers zegt "Ga open source doen", en je hebt bedrijven waar mensen werken die uit eigen beweging met communities bezig zijn. Voor die laatste veranderd er niks. Maar als je het als werknemer als opdracht krijgt, dan doen ze het gewoon omdat ze er voor betaald krijgen. En dan moet je kijken waarom dat bedrijf bezig is met communities. Maar dat is vaak gewoon geld verdienen, PR, markering, mensen opleiden ergens heel goed in te worden. Bijvoorbeeld Yahoo heeft ook mensen in dienst die werken aan Apache. Zij moeten dingen dingen maken, en daarvoor hebben ze de community nodig. Daarvoor moeten

deze mensen ook de goede contacten binnen de community hebben.
Maar als je het hebt over de overheid, dan blijft het iets wat de individuele ambtenaar wel moet willen doen. Als je bijvoorbeeld mensen die bezig zijn met financiele administratie bij elkaar wilt brengen om na te denken over de software, dan kun je beter daar geld tegen aan gooien, en zeggen dat ze vier uur per week daar gezamenlijk over na moeten denken. Als mensen communities binnen de overheid willen opzetten, dan zal het wel zo moeten zijn dat zij aan het einde van het jaar moeten kunnen laten zien wat voor resultaat het heeft gehad. Als een werknemer het echter gewoon leuk vindt, en het naast zijn reguliere werkzaamheden doet is het verder geen probleem.

Daniel Communities zijn nu vooral virtueel. Wat zou er veranderen als veel niet meer virtueel gaat? Bijvoorbeeld bij veel projecten zijn er stuurgroepen die face-to-face overleggen over de richting van het project.
Ruben Je moet twee verschillende zaken onderscheiden. Ten eerste heb je de maat van organisatie; stuurgroepen, werkgroepen, allemaal een soort bureaucratisering. En ten tweede heb je elkaar echt fysiek zien. Elke community ontmoet elkaar wel een aantal keren per jaar, bijvoorbeeld op congressen. Er zijn ontwikkelaars die elkaar nog nooit hebben gezien, maar de meeste zien de andere ontwikkelaars geregeld. Dat is natuurlijk geen enkel probleem, het maakt het misschien juist makkelijker. De mate van organisatie is iets heel anders. Vaak worden de argumenten gebruikt dat er sturing of een richting nodig is, bovendien doet open source dat zelf ook, die hebben een stichting of zo. Maar de wijze waarop organisaties iets willen organiseren is heel anders dan de wijze waarop de open source community zich organiseerd. Ik weet zeker dat een stuurgroep van bijvoorbeeld DigiD gaat zeggen hoe DigiD er volgend jaar uit moet zien, en als er iets anders ontwikkeld wordt is dat niet goed. Maar bij open source is het juist dat je je wensen op een website zet, maar als er iets anders ontwikkeld wordt is dat ook prima. Als het niet precies bij de wensen past, dan wordt gewoon gekeken wat ze er wel mee kunnen. Maar bij zo'n stuurgroep, daar zitten dan vaak de belangrijke mensen in, en als die aan de werkgroep vertellen wat er moet gebeuren is het het einde van het project als open source.
Daniel Formele stuurgroepen zijn dus echt dodelijk voor open source communities?
Ruben Als de werkgroep gewoon betaald krijgt om uit de voeren wat de stuurgroep zegt, dan niet, maar dan is het geen community, en moet je het ook niet zo noemen. Dat is dat gewoon projectontwikkeling. Maar als een stuurgroep gaat vertellen wat andere mensen vrijwillig moeten gaan doen, dat gaat nooit werken.
Een stuurgroep kan echter wel zeggen dat de funding beschikbaar stellen voor zaken die ze echt belangrijk vinden. De rest van de community kan dan blijven doen wat ze willen, en er zullen dan ook ontwikkelaars zijn die die opdracht willen uitvoeren om het geld te krijgen. Wat er dan nog vaak gebeurd is dat er wordt gezegd: deze persoon gaat het doen er krijgt daar een bedrag voor. Dat geeft ook problemen, want de ontwikkelaar die het uitvoert kan met meerdere dingen bezig zijn. Hierdoor kan betaalde project vertraging oplopen. Als andere ontwikkelaars dan afhankelijk zijn wat van die ene ontwikkelaar had moeten opleveren, kunnen zij ook niet verder, en ze kunnen het ook niet zelf implementeren, wat dan krijgen ze er niet voor betaald. Als er dus funding is moeten dat worden gegeven aan de eerste persoon die het oplevert, zonder het van te voren aan iemand te gunnen. Ontwikkelaars krijgen reputatie op wat ze ontwikkeld hebben, niet omdat ze ergens aan bezig zijn. Dus als een community werkt op basis van vrijwilligheid moet je mensen niet van te voren gaan betalen om dingen te gaan uitvoeren.
Daniel Er zijn ook projecten waar er twee soorten communities ontstaan, een voor de managers, en een voor de programmeurs. Kun je dan nog spreken van open source?
Ruben Open source zegt natuurlijk alleen iets over de licentie. En zo'n project kan natuurlijk best een open source licentie hebben. Maar je kunt dan moeilijk meer spreken over communities. Bij te veel sturing is kan er gewoon geen spraken zijn van communities. Maar ik kan me ook heel goed voorstellen dat de noodzaak tot sturing bij sommige projecten veel groter is dan bij andere, omdat je zeker wilt weten wat de uitkomst is. De vraag blijft natuurlijk altijd wat je doet met zaken die niet direct bij de doelstelling horen. Het kan best zo zijn dat iets eerst vrij nutteloos lijkt, maar dat er later toch iets heel handigs uit komt.
Daniel Is sturing dan nooit goed?
Ruben Je kunt sturen op een aantal zaken. Je kunt sturen op wie er mee mag doen, de meeste open source communities zijn daar heel erg open in. Je kunt ook sturen in wie mag ontwikkelen. Je kunt ook sturen in de technische specificaties, in de eisen die het heeft. Je kunt ook sturen in het technisch ontwerp. Je mag dan van alles ontwikkelen, maar het moet wel aan het technisch ontwerp voldoen. Je kunt ook sturen in de interfaces. Ik denk echt dat sturing de crux is voor open source. Elke community stuurt, de vraag is alleen: waar stuur je op. Maar in de meeste organisaties wordt gestuurd op echt alles, en dat is niet goed. Het enige waarop een organisatie zeker moet sturen is welke versie van de software wordt gebruikt, niet dat sommige gebruikers oudere versies of ontwikkelingsversies gebruiken. Maar als mensen in hun vrije tijd andere dingen willen testen moet je dat vooral toelaten.

Daniel In je boek beschrijf je het ui-model, met ringen van verschillende betrokkenheid. Daarbij ontwikkelen de programmeurs in de eerste plaats voor zich zelf, en andere gebruikers kunnen de software gebruiken en er over meedenken. Wat je denk ik bij veel organisaties hebt is dat de ontwikkelaars hele andere mensen zijn dan de uiteindelijke gebruikers. Ontwikkelaars ontwikkelen dus niet meer uit eigen motivatie.
Ruben Wat denk ik een groot probleem is bij veel bedrijven is, precies zoals je beschrijft, dat er een ontwikkelafdeling is met programmeurs, en een gebruiksafdeling met gebruikers. In veel organisaties heb je dan een klankbordgroep, die commentaar levert, dan ontstaat er feedback en worden er dingen verbeterd. Dat is heel formeel. Open source doet eigenlijk hetzelfde, met veel kortere release-cycles, ze laten alles op hun website zien en elke gebruiker kan het testen. Dat is bijna dezelfde manier van organiseren, waarbij de afstand tussen gebruiker en ontwikkelaar klein is.
Daniel Maar dat gaat er wel van uit dat gebruikers geïnteresseerd zijn in de software. Ik kan me voorstellen dat als je een open source pakket op de financiële administratie hebt, dat gebruikers niet erg geïnteresseerd zijn in de software, maar gewoon hun werk moeten doen.
Ruben Zo'n houding zou eigenlijk al fout zijn. We gaan denk ik steeds meer toe naar een situatie waarin mensen professional worden, en verantwoordelijk zijn voor hun werkzaamheden. En dat ze dus ook kunnen bepalen hoe ze iets uitvoeren. Dan kunnen ze uiteindelijk beter hun werk uitvoeren.
Daniel Maar bij een financiële administratie ben je toch ondersteunend voor iemand anders, dan heb je toch vrij weinig keuzen?
Ruben Ja, maar zij weten dan ook het beste met welke regels ze te maken hebben, en hoe zo'n systeem er uit zou moeten zien. Maar er zullen binnen elke organisatie natuurlijk altijd mensen zijn die minder gedreven zijn en alleen maar hun tijd tot hun pensioen willen uitzitten. Maar ik denk dat in het overgrote deel van de organisaties er genoeg mensen zijn die hele slimme ideeën hebben over hun zij hun werk moeten doen. Als je mensen gewoon nieuwe software geeft met de opdracht die te gaan gebruiken zullen ze daar misschien weinig zin in hebben, maar als je ze inspraak geeft dan weer ik zeker dat mensen veel gemotiveerder zijn. Of je het model dan nog steeds moet zien als ringen, of als iets met een klankbordgroep maakt niet zo veel uit. Het enige probleem met een klanbordgroep is dat er in hun rol bijna vast ligt dat ze wat kunnen roepen, achterover kunnen leunen en verder wel zien wat er uit komt. Terwijl als mensen echt medeverantwoordelijkheid krijgen, dan gaat dat veel beter. Veel bedrijven outsourcen hun software. Ze stellen wat eisen op, besteden het aan, en dan mag de goedkoopste het uitvoeren. Veel bedrijven komen daar nu weer van terug, en willen een veel duurzamere relatie, waarbij samenwerking veel meer centraal staan. Dat lijkt mij de goede manier om software te ontwikkelen, en dat lijkt al veel meer op een community.
Daniel Die eigen invloed kan natuurlijk ook weer problemen opleveren. Het zou zo kunnen zijn dat de systeembeheerders het niet echt leuk vinden als medewerkers wat gaan experimenteren met software en af en toe een nieuwe versie installeren.

Ruben Geen enkel bedrijf dat in open source actief is heel elke dag andere versies van programma's geïnstalleerd staan. Je hebt een stabiele versie die iedereen gebruikt, en een ontwikkelingsversie. Hieraan denken bijvoorbeeld een keer per week medewerkers mee over de richting. En wanneer de stabiele versie moeten worden geüpdatet, dat is een beslissing voor het management.
Daniel Maar dan is het schema van “release often” en veel mensen die fouten kunnen ontdekken niet meer geldig. Alleen de ontwikkelafdeling ziet dan de ontwikkelversie echt veel, de eindgebruikers – die de meeste kennis hebben – nauwelijks.
Ruben Maar op dit moment zijn de meeste gebruikte open source applicatie die applicaties waarbij de systeembeheer de gebruiker is. En als er software is voor de eindgebruikers dan kan dat nog steeds getest worden. Iemand van de administratie zal dan eens kijken of hij met die nieuwe software nog steeds dezelfde dingen kan doen, maar zal zeker niet direct de hele administratie overzetten.

Daniel Een van de laatste conclusies in je boek is: communities zijn niet maakbaar. In mijn onderzoek wil ik uiteindelijk beleidsadviezen geven om juist communities op te zetten. Kan dat?
Ruben Ja, communities zijn zeker goed op te zetten. Het idee van zelf-organisatie van communities is ook vrij stellig geformuleerd, om het idee onderuit te halen dat je zomaar alles kunt maken. Je kunt alleen niet van te voren bepalen waar het heen gaat, en of het een succes wordt. Je kunt een aantal dingen doen om te zorgen dat mensen enthousiast raken. Dat kan door geld, minder streng te zijn op sturing. Je moet ook zorgen dat er al iets ligt, dat mensen wat zien waarover ze enthousiast kunnen worden. Je moet ook goed kijken wat je ontwikkeld. Je kunt een nieuw besturingssysteem ontwikkelen, waar er al heen veel van zijn, en waar iedereen zijn voorkeur al heeft vastgesteld. Dat zal waarschijnlijk niet zo'n succes worden. Je kunt ook iets ontwikkelen waar nog maar een concurrent is, dan heb je veel meer kans dat het slaagt.