How to buy something that is free?

developing grounded design principles from theory and practice, for the inclusion of free/libre open source software in an organization's tactical procurement of software

Verheesen, M.P.M.W.

Award date:
2010

Eindhoven, June 2010

How to buy something that is free?

Developing grounded design principles from theory and practice, for the inclusion of free/libre open source software in an organization's tactical procurement of software.

by
M.P.M.W. Verheesen

BASc Electrical Engineering - Fontys 2007
Student identity number 0531934

in partial fulfilment of the requirements for the degree of

Master of Science
in Innovation Management

Supervisors:
dr.ir. I.M.M.J. Reymen
dr. ir. J.J. Berends

TUE. Department of Industrial Engineering and Innovation Science
Series Master Theses Innovation Management

Subject headings: (F/L)OSS, procurement, adoption

Technische Universiteit
Eindhoven
University of Technology

Department of Industrial Engineering and Innovation Science

Group of Innovation, Technology Entrepreneurship and Marketing
Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
http://w3.ieis.tue.nl/en/groups/item/

Author
BASc. M.P.M.W. Verheesen

Mentor
dr.ir. I.M.M.J. Reymen

Second assessor
dr. ir. J.J. Berends

Keywords
(F/L)OSS research

Date
June 7, 2010

Version
1.9

How to buy something that is free?

Developing grounded design principles from theory and practice, for the inclusion of free/libre open source software in an organization's tactical procurement of software.

Abstract

This master thesis generates new knowledge about the 'procurement' of (F/L)OSS by government organizations. And develops design principles that would help government organizations (re)design their 'procurement' policies to allow for (F/L)OSS. In order to achieve these goals the following research questions were posed: how do government organizations deal with (F/L)OSS, what are the challenges and how can the 'procuring' be improved. These research questions were answered by formulating a set of 7 design principles grounded in theory and practice. These principles form guidelines that can be used to create solutions for how to deal with (F/L)OSS. Developing these design principles also generated new knowledge about the 'procurement' of (F/L)OSS by government organizations.

Management summary

The research in this thesis is on the procurement of (F/L)OSS (Free/Libre Open Source Software) by government organizations. The main research question is how do (government) organizations deal with (F/L)OSS when procuring software.

Background information

Characteristics of (F/L)OSS

There are two opposite types of software in the world, proprietary software *products*[1] and (F/L)OS-Software *programs*. Users of the latter type of software are allowed to use, alter and redistribute (share) the software (Coar, 2006). This is in contrast with proprietary software products where users are reduced in these rights by a license agreement (for instance an EULA[2](Microsoft, 2009; Wikipeadia, 2009)).

The implication of the existence of (F/L)OSS is the creation of two paradigms. The *product* and the *program* paradigm. In the former paradigm, software is seen as a product, sold by proprietary vendors and in the latter software is seen as highly codified knowledge, created by communities of developers working together on the Internet.

Motive for research

Because organizations are used to procuring *products*, they seem to be having problems "procuring" (F/L)OSS *programs*. The procurement practices that are utilized by organizations, are incapable of handling the characteristics of (F/L)OSS. Furthermore, current research has not addressed this situation. There is research that explains why (F/L)OSS is being used. (F/L)OSS adoption factors and barriers are the focus of that research. And there is research done on procurement, which argues why a preference for (F/L)OSS should be made and how to formulate such a preference. However, none of the prior research seems to consider *how* organizations should "procure" (F/L)OSS, taking into account the organizational side of the story as well as the implications for the existing procurement processes.

[1] One can argue whether software is highly codified knowledge or a product (Hippel & Krogh, 2003; Perens, 1999; Bonaccorsi & Rossi, 2003).

[2] This legal contract between the user and the vendor of the software prohibits the users from altering, sharing and using the software for any purpose

Research goal

This research aimed to close this research gap. It therefore had two goals. Goal one was to generate new knowledge about (F/L)OSS "procurement" and the second goal was to develop design principles that will help organizations (re)design their procurement policies to include (F/L)OSS. Based on these objectives, three research questions can be defined:

- How do organizations deal with (F/L)OSS programs when procuring software for their operational processes?
- What problems are found in these organizations when acquiring (F/L)OSS programs and have they found solutions for this?
- How can the "procuring" be improved?

Research method

In order to answer these questions, a research model has been developed. In this research model knowledge from the literature is combined with knowledge from the field of practice. This is done in three steps. First by creating practice-based design principles from the field practice and second by creating theory-based design principles from the literature. In the third step, these two sets of design principles were synthesized to create a final set of design principles grounded in theory and practice (see figure 1).

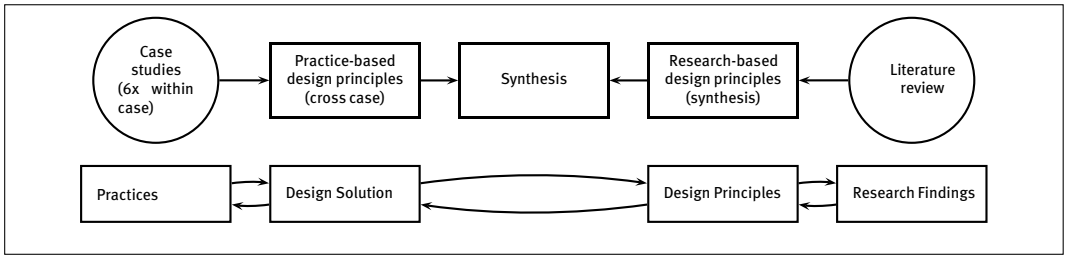

Figure 1: Schematic of the research design adopted from De Jager (2009) and Burg et al. (2008).

The theory-based design principles have been extracted from the literature by using a systematic literature review and synthesis. And the practice-based design principles have been extracted by making use of a multiple case-study.

Research findings

In total seven final design principles have been created by synthesizing eight practice-based design principles and eleven theory-based design principles. The final design principles are formulated in CIMO logic. CIMO logic is a way to formulate and display design principles. Design principles have the goal to explain why (mechanism) acting in a particular way (an intervention) in a certain environment (the context) will cause an outcome. In the following boxes the seven final design principles are summarized.

Summary final design principle 1: acquiring (F/L)OSS	
In the context: of a (strategic) choice for (F/L)OSS use	
Outcome: to implement (F/L)OSS	
Intervention	*Mechanisms*
By downloading, installing and configuring (F/L)OSS yourself, by following one of three strategies depending on the context, (a) business-critical: gradually evolve the (F/L)OSS solution in parallel with the existing system, until the system meets the requirements. Then start migrating users when the (F/L)OSS solution is mature and has proven itself. (b) non-critical: just download, instal and configure the (F/L)OSS program. (c) introducing new ICT functionality: experiment with different (F/L)OSS alternatives then lobby for organizational support. (d) always: plan the project extensively.	Because if there is knowledge of (F/L)OSS, implementing it lies within the capability of the IT-staff. (a) evolving the system in parallel requires beta releasing to users which builds organizational support and minimizes risk by demonstrating the solutions capability. (b) a non-critical context the implementation can be done with slack resources and the entrepreneurial spirit of the IT-staff will find a (F/L)OSS solution. (c) because the functionality does not exist yet, so it won't be missed this reduce risk. After completion, for easy adoption, lobbying and change management is required. (d) because factors like what to change, when, technical and interconnections issues, scope, necessary resources must not be underestimated.
By procuring a (F/L)OSS *solution* from a system integrator, by using a procurement process which uses specific terminology in the requirements to specify the (F/L)OSS characteristics.	Because there might not be sufficient knowledge about (F/L)OSS or the organization wishes to outsource IT then procuring integration services works. Specifying neutral requirements and using terminology based on the (F/L)OSS characteristics works because it will request (F/L)OSS without asking for it specifically.
By procuring the support of a (F/L)OSS knowledge specialist by using a procurement process which uses specific terminology in the requirements to specify the (F/L)OSS characteristics. This knowledge specialist will assist with the downloading, installing and configuring of the (F/L)OSS program or solution.	Because there might not be enough internal knowledge about (F/L)OSS but the organization wants to internalize this knowledge in order to become self (reliant) and proficient with (F/L)OSS. Specifying neutral requirements and using terminology based on the (F/L)OSS characteristics works because it will request (F/L)OSS without asking for it specifically.

Summary final design principle 2 : entrepreneurship	
In the context: of adoption of (F/L)OSS by downloading and implementing (F/L)OSS yourself	
Outcome: stimulating the individual entrepreneurial attitude towards (F/L)OSS of employees and obtaining personal effort, employee commitment and triggering adoption can be done	
Intervention	*Mechanisms*
by stating a precise definition of strategic intent for instance by pointing out the national policy and (F/L)OSS philosophy, but being flexible in means of how to reach this goal.	this will trigger employees to get committed and start searching for opportunities to use (F/L)OSS, because current resources do not suffice, innovation is necessary.
by organizing for a high perceived capability for solutions, by giving employees room to experiment and educate employees in different kinds of (F/L)OSS alternatives by monitoring (F/L)OSS repositories, hiring (F/L)OSS experts, sending key employees to conferences and obtain subscriptions to trade magazines.	By creating a high perceived capability for solutions the organization makes sure the employees also know what to do in order to reach the strategic goal. This is very important in cases were there might not already be compatible (F/L)OSS knowledge at the IT-staff. This has to be created first, before entrepreneurial efforts can emerge.

Summary final design principle 3 : top management support	
In the context: of radical, high-risk initiatives such as acquisition of complex FLOS-Software in a government setting with incomplete information about the decision	
Outcome: obtaining necessary resources such as time, space, equipment, people and communicating, supporting and sharing, vision and goals is done	
Intervention	*Mechanisms*
by creating top management support. This is done by changing the perceptions that managers have of (F/L)OSS and by providing them with information about (F/L)OSS. This can be done by pointing out a choice for (F/L)OSS would comply with national policy, show a successful proof of concept (poc) and/or business case. And by making cost calculations. And in these calculations account for the switching costs when vendor lock-in is deepened by opting for a proprietary alternative.	Knowledge is gained therefore the urge to imitate the decisions of other managers is reduced. Taking away irrational fear of damage to the managers reputation, will cause them to decide more rationally. Therefore the presented reasons will get a fair evaluation. Lower costs argument will have impact because public money spent. Successful poc and business case shows confidence the solution works. Only top management has the legitimacy and power to distribute the resources.

Summary final design principle 4 : lock-in	
In the context: of technical compatibility issues (vendor lock-in) with legacy technology	
Outcome: reducing the lock-in is done	
Intervention	*Mechanisms*
by convincing the existing vendors to use open standards.	because open standards make it possible to interchange parts of a total IT solution
by replacing the legacy systems by installing (F/L)OSS (alternatives) as much as possible	because the characteristics of (F/L)OSS make vendor lock-in impossible. (F/L)OSS uses open standards and thus makes it possible to interchange parts. And by using (F/L)OSS one does not rely on a specific vendor but instead can get support from many different companies.
by encapsulating the parts that cannot be replaced, by using middleware and virtualisation techniques.	because these will make it possible to limit the amount of the legacy software and thus increasing the opportunity to use (F/L)OSS.
by preventing that current investments will make future investments of (F/L)OSS more difficult	because lock-in will at least not be increased by the new investments. This will also prevent these current investments to become sunken costs in the event that (F/L)OSS is chosen in the future.

Summary final design principle 5 : architecture	
In the context: of acquiring (F/L)OSS	
Outcome: implementing an architectural IT policy will structure the strategic application of ICT and protecting (F/L)OSS investments can be done,	
Intervention	*Mechanisms*
by creating an IT architecture policy based on (F/L)OSS and open standards guarded by top management and supporting processes in the organization	because defining the basic principles of (F/L)OSS that are in line with the strategy of the organization in a formal policy at a high level in the organization will put a price on non compliance to the architecture and thus will make sure employees have the necessary motivation to create room in projects or procurement's and thus choose technology that is strategically beneficial to the organization and will make it difficult to argue for proprietary systems that are not in the best strategic interest of the organization.

Summary final design principle 6 : organizational support	
In the context: of downloading and implementing (F/L)OSS yourself	
Outcome: creating organizational support can be done	
Intervention	*Mechanisms*
by reducing misunderstanding by explaining what (F/L)OSS is and informing employees about the possibilities by monitoring (F/L)OSS repositories, hiring (F/L)OSS experts, send key employees to conferences and subscribe to trade magazines.	Because this intervention brings external knowledge into the organization and will therefore show employees what (F/L)OSS exists and make them understand and familiar with it.
by increasing trialability by letting end-users play with the solution. This can be done because it's easy to setup a working pilot with (F/L)OSS programs and letting end-users make the final decision.	because having end-users experience the solution and make the final decision will enable the possibility to test if the solution fulfills the requirements, will give end-users a feeling of taking part in the project and that their wishes are being heard. This generates internal support for the final solution.
by communicating continuously with the end-users by showing that the solution creates benefits for them and sharing experiences and results. This increases observability and takes away fear.	because the visibility of the results of the technology, will explain the situation, reduce fear of the solution and show the goals. And thus the more likely the technology will be adopted.
by training end-users in the new technology.	because training will, increase the usability of the (F/L)OS-software, will show that the solution works and thus reduce uncertainty and doubt about the technology.

Summary final design principle 7	
In the context: of acquiring (F/L)OSS	
Outcome: creating an environment ready for (F/L)OSS adoption is done:	
Intervention	*Mechanisms*
by adjusting work habits.	because the new system might require different ways of performing tasks or offers more efficient ways of performing these tasks, adjustment of work habits might be necessary.
by adjusting procedures to accommodate (F/L)OSS procurement.	because current procedures might not be compatible with certain characteristics of (F/L)OSS. Therefore adjusting the procedures by taking into account these characteristics will help (F/L)OSS procurement.
by adjusting existing roles.	because the tasks performed in the roles can be the opposite of what tasks are required to introduce (F/L)OSS. Changing the tasks of a role with (F/L)OSS procurement in mind will make sure people are free within their roles to procure (F/L)OSS.

Implications

The first contribution of this research is the insight (shown by final design principle 1) that (F/L)OSS cannot be procured. But it is possible to procure integration services and (F/L)OSS knowledge. The second contribution are two strategies to implement (F/L)OSS in an organization, that emerged during this research. These are "building niches" and the "coordinated effort" approach.

In the cases groups of ambitious people with compatible (F/L)OSS knowledge are implementing (F/L)OSS themselves. These initiatives form niches of (F/L)OSS. Although this works, it leaves the final implemented (F/L)OSS solution vulnerable to opportunistic behavior.

Investments in (F/L)OSS can be protected by a coordinated effort to (F/L)OSS implementation. In this approach, strategic (or even political) objectives are formulated at the highest level in the organization, on the basis of business goals. These strategic objectives are then translated into an IT architecture which is guarded by top management. This way, opportunistic behavior becomes more difficult.

The third contribution are the remaining design principles that can be used to create an environment which makes (F/L)OSS adoption easier.

Limitations

There are three main limitations in this research. First of all the field that was investigated is relatively new and apart from the adoption framework, no theoretical framework existed to guide the research, or that the research could test.

Second the generalizability of the research is limited. Only six cases were investigated and results are bound to these cases. There are however two arguments that indicate that the results can be applied broader. First, the stories from the cases appear very similar to other cases in the same context. Secondly, the culture of the people dealing with the issues mentioned in this research is widespread.

A third limitation is that the design principle have only been designed and not tested. It is not proven that the design principles interventions function correctly and that the mechanisms are precisely understood. These three limitations cause this research to be of a highly exploratory nature and further research is necessary.

Preface and acknowledgement

This report is the result of my master thesis project, performed in partial fulfilment of the requirements for the degree of Master of Science in Innovation Management at the Eindhoven University of Technology. During my master studies, I noticed how a lot of theoretical organizational concepts that solved problems, looked similar to what I'm used from (F/L)OSS communities. It also appeared to me that some organizations have great difficulty dealing with (F/L)OSS. To me this was baffling. Being an electrical engineer, I'm used to applying (F/L)OSS in practice. In my opinion organizations can benefit from understanding the (F/L)OSS principles. This insight motivated me to execute this master thesis project and hopefully enable organizations to deal with (F/L)OSS.

It soon became apparent that yet again in my life, I was taking the road less travelled. As illustrated by this quote on my first interview: "Open source and procurement is a seriously difficult thing, in that regard you chose the perfect subject" (Sander Mittertreiner, DPO). Luckily I received a lot of help, without it, this work would not have been possible. In this place I would like to thank them.

First of all, I'm greatly indebted to Isabelle Reymen my mentor. Who indulged my sometimes wild ideas and gave me the necessary freedom to let them become the text that lies before you. She was able to funnel all my energy and enthusiasm spent on random ideas, towards a goal unknown to us at the beginning. And I would like to thank Hans Berends, my second mentor, who's impeccable logical reasoning proved enlightening in all occasions.

I also want to thank Arjen Kamphuis. For supporting the research and showing me around in the rich and diverse open source community of the Netherlands. I also enjoyed the brainstorming sessions and I'm still amazed by his skill to reduce a complex issue to a single comprehensive sentence, a truly remarkable capability.

Then special thanks to Lucas van Luijtelaar for our telephone talks which prevented us from loosing our minds over all the work. And Sanne van der Burg for coming to the presentation ;)

Furthermore thanks go to: Annie Machon, for help with the English language. The people that helped me by reviewing the drafts and my methods. The people I've met and who were so willing to cooperate, especially the interviewees, because without them this work would not have been possible at all.

And finally I would like to thank my parents for their everlasting support and Sina, my girlfriend. For her love, patience and encouragement during the many lonely hours (often late at night) of typing this thesis report.

Maurice Verheesen

Eindhoven, June 2010

List of Figures

List of Tables

Table of contents

Table of contents

Table of contents

Title
How to buy something that is free?

1 Introduction

The research in this thesis is on the procurement of (F/L)OSS (Free/Libre Open Source Software) by government organizations. The main research question is how do (government) organizations deal with (F/L)OSS when procuring software. This introduction will provide the reader with the necessary background information. (F/L)OSS will be explained first then an overview is given of the usual process that organizations follow when procuring software and finally the problem will be shown when trying to "procure" (F/L)OSS.

There are two opposite types of software in the world, proprietary software *products*[1] and FLOSS-Software *programs*. Users of the latter type of software are allowed to use, alter and redistribute (share) the software (Coar, 2006). This is in contrast with proprietary software products where users are reduced in these rights by a license agreement (for instance an EULA[2](Microsoft, 2009; Wikipeadia, 2009)). Because (F/L)OSS does not impose on any of the rights to use, alter and redistribute the software, is the opposite of proprietary software.

These rights are put into practice by two things. A license that protects the rights to use, alter and redistribute. And access to the software's source code[3] is required, in order to be able to alter the behavior of the software and to redistribute it. This explains the name "open source", since it's the *source code* of a program that is *opened* and can thus be shared *freely*. This is the opposite of proprietary software products, because vendors of those products see the source code as a company secret that must be closely guarded instead of shared[4].

(F/L)OSS programs are created by different kinds of people working together in a project on the Internet. The people working in these projects can for instance be hobbyists, professionals sponsored by companies and users providing usability feedback and bug reports. They all have various reasons for helping in the project and/or in the development of the program. This online activity forms the production environment in which these (F/L)OSS programs are created and is the delivery method of (F/L)OSS programs into the "software market" where they compete with proprietary software products. When the development on a new release of a (F/L)OSS program is finished[5], it is distributed and can thus be used by for instance organizations.

Organizations procure products by making use of a procurement process. They do this, because the

[1] One can argue whether software is highly codified knowledge or a product (Hippel & Krogh, 2003; Perens, 1999; Bonaccorsi & Rossi, 2003).

[2] This legal contract between the user and the vendor of the software prohibits the users from altering, sharing and using the software for any purpose

[3] Think of the software source code as the building blocks of the software program. An analogy of source code is a recipe of your favorite dish. With the source code it is possible to reproduce (build) the program, just as it is possible to prepare your favorite dish with the help of the recipe.

[4] Compare for instance with Microsoft Windows. The EULA forbids the redistribution (sharing) of this operating system. In fact, Windows is not owned by the customers at all. Customers buy a right to use. c.f. Microsoft (2009))

[5] Notice the agile software development method principles used by (F/L)OSS here. Those principles regard software as never finished. Because there is always need for improvement, bug fixes or implementing changed customer requirements (Brooks, 1995; Boehm, 1981; Baetjer, 1998). This is counter to the waterfall development model that is mostly used in the product paradigm which assumes that software can actually be delivered as a finished product.

organizations management wants some level of control over the procurement of products used by the organization. This control is necessary mainly for three reasons (Weele, 2005), first control over expenditure (e.g. prevent buying products that are not needed), second they want to "standardize business processes by managing contacts with vendors and establishing effective links that enable continuous improvement of vendor performance" (e.g. prevent buying from different vendors) and thirdly manage the contracts formed with vendors to enable "on time delivery at the right quality and quantity at the lowest price" (Weele, 2005).

Proprietary software product vendors have adapted this product paradigm. The vendors sell a software *product* defined by a licence. The product has predefined functionality that a customer is not allowed to change and the product has a life cycle (even an end-of-life) just as normal products. Contracts can be made with the vendor about delivery, quality and "quantity" of the product. A part of these contracts are formed by, the software license agreements. And just as the normal contracts with other vendors, these licences need to be managed too. Because licences management helps proof to the proprietary software product vendors that the organization does not infringe on any software licences' terms (i.e. run the software illegally) (BSA, n.d.).

On the other hand there are (F/L)OSS *programs*, which is software that is free (*gratis* and can also be *libre*). This software comes from an online project, not from a vendor. That means it's hard to make contracts or "establish links" in the traditional sense. Furthermore (F/L)OSS *programs* can be easily altered to fit a particular requirement. This is the opposite of the *product* paradigm where products are sold " as is". And from expenditure point of view (F/L)OSS is different, because (F/L)OSS programs can be downloaded and used *gratis*. So by definition (F/L)OSS is incompatible with the procurement process.

But some aspects of the procuring process are relevant. Not all (F/L)OSS licenses are the same and they still are legal binding documents with restrictions that need to be followed. Furthermore support on (F/L)OSS is procured separately. The fact that one can alter the software, causes strategic questions that need to be answered. Also the history and philosophy of (F/L)OSS is radically different from the product paradigm, so IT-staff will require training to deal with this. All of these issues need to be addressed by an organization.

Scientific research has been done on some of the challenges facing an organization that starts using (F/L)OSS. There has been research done on reasons and barriers for the adoption of (F/L)OSS. These articles state the organizational antecedents when implementing (F/L)OSS and thus answer why (F/L)OSS was implemented in the organization in question, but do not mention how this affects the procurement process. And there have also been studies to the "procurement" of (F/L)OSS. They focus mainly on what to do when an outside vendor supplies a particular (F/L)OSS program. Without paying attention to the fact that a (F/L)OSS program can be altered and what kind of issues and possibilities this generates in terms of requirements[6]. This literature about procuring simply mentions the other possibility that the (F/L)OSS program can be downloaded, largely ignoring what kind of organizational change will be necessary when the (F/L)OSS program is "procured" in this way. Ignoring the issues in this way, will not resolve them. So it is obvious there is a gap in the literature.

To put it more strongly, it's hard to find a word[7] that covers what is being done here. It is not buying, because (F/L)OSS programs costs no money. It is not procuring, because (F/L)OSS programs have no vendor. It is not using, because this term does not entail making a choice and would thus limit the scope. It is not implementing, because that word refers to the technical actions before using

[6] For instance downloading a (F/L)OSS program that fulfils 90% of the requirements and building by contract the final 10%.

[7] Since something needs to change in the procurement process, "procurement" in quotes or the word 'acquiring' is used in the rest of the thesis.

the program and after the choice has been made. And finally it is not adoption, because adoption only looks at reasons and barriers *why* to use and not *how*).

That is why this research will contribute to the literature by trying to answer the question how do (government) organizations deal with (F/L)OSS, when procuring software. The goal is to fill the mentioned gap by formulating design principles that combines solutions created in the field of practice, with existing fragments from the literature about adoption, procurement and implementation of (F/L)OSS. These design principles can then be used to create solutions that enable organizations to deal with (F/L)OSS.

Formulating the design principles is done by integrating the knowledge of these three kinds of research. First; design principles from the scientific literature about adoption and procurement of (F/L)OSS are induced (chapter 4.1, secondly; the solutions created in the field of practice are extracted in a multiple case study (chapter 4.2) and thirdly; by synthesizing these two sets of design principles (chapter 4.3) into a final set of design principles grounded in research and practice.

The provocative title of the thesis "How to buy something that is free[8]?" basically summarizes the subject very well. This question looks like a *contradictio in terminis*. To buy means "to acquire the ownership of (property) by giving an accepted price" (Webster, 1913). Free has two definitions; first the monetary value of zero or "gratis"; and second, freedom, as in "exempt from subjection to the will of others" (Webster, 1913). Both definitions of free do not fit with buying. Because giving the accepted price of zero renders the term useless. Moreover something that is exempt from subjection to the will of others (as in the case of (F/L)OSS) cannot be owned in the first place and thus can also not be acquired. So the statement seems self-evidently false. But as was made clear in this introduction, when dealing with (F/L)OSS in an organizational setting, the statement becomes a valid practical question. This is because of the way organizations are used to handle software acquisition and the special characteristics of (F/L)OSS that are not compatible with this process. The contribution of this research is the set of design principles that will guide (government) organizations that embark on the paradigm-crossing voyage of how to buy something that is free.

[8] The question was mentioned by both Johan de Man and Sander Mittertreiner in their interviews.

2 Motivation and research questions

In this section the motivation for the research is explained and the research questions are developed. The motivation for this research is shown through a confrontation of (F/L)OSS literature, current news articles and software procurement practices. This will illustrate why the traditional view on procurement does not work for (F/L)OSS. The inconsistencies form the arguments for the research questions.

First a gap in literature is exposed by giving a short literature overview[1]. Then issues with three procurement practices are revealed that illustrate the point that a conventional procurement process does not work for (F/L)OSS. From this the research questions are formulated. And finally an overview of the thesis report is given, as a reading guide.

2.1 Literature gap

There are four categories of literature about (F/L)OSS. Literature that discusses motivation to develop (F/L)OSS (Lakhani & Wolf, 2005; Hippel & Krogh, 2003; Krogh, Spaeth, & Lakhani, 2003; Lerner & Tirole, 2002), articles about the (F/L)OSS development process (Woods & Guliani, 2005; Crowston, Li, Wei, Eseryel, & Howison, 2007; Markus, 2007; Mockus, Fielding, & Herbsleb, 2006), then adoption of (F/L)OSS and finally procurement of (F/L)OSS.

The literature about adoption of (F/L)OSS, discusses the reasons why (F/L)OSS is adopted. For instance why governments should adopt (F/L)OSS (Applewhite, 2003; Lessig, 2002; J. Lee, 2006; Valimaki, Oksanen, & Laine, 2005; Comino & Manenti, 2005; Hahn, 2002), why software vendors should use (F/L)OSS (Hauge, Sørensen, & Conradi, 2008; Mathisen, 2008), why perceptions influence (F/L)OSS adoption (Morgan & Finnegan, 2007; Ozel, Jovanovic, Oba, & Leeuwen, 2007; Holck, Larsen, & Pedersen, 2005), why businesses adopt (F/L)OSS (Dedrick & West, 2004; Nagy, Yassin, & Bhattacherjee, 2010; Glynn, Fitzgerald, & Exton, 2005), why one particular (F/L)OSS program was adopted (Varian & Shapiro, 2003; K. Ven, Van Nuffel, & Verelst, n.d.), or why multiple programs were adopted (Pare, Wybo, & Delannoy, 2009; Fitzgerald & Kenny, n.d.; Munoz-Cornejo, Seaman, & Koru, 2008), or (F/L)OSS adoption across countries (Cassell, 2008; Hwang, 2005). This explains why adoption happens, not how it happens. None of these articles link adoption to procurement for instance. But in the practical field, people are used to *procuring* software products.

There is a small and very recent body of literature about "procuring" (F/L)OSS by governments (Ghosh, Krieger, Glott, & Robles, 2002; Ghosh, 2010; Eilander, 2008) and SME's[2] (Daffara, 2008). However this literature does not account for the required organizational change or the barriers to adoption. It only focusses on the formulation of the (F/L)OSS characteristics in public tenders (Ghosh, 2010; Eilander, 2008) or risks, myths and business models (Daffara, 2008).

So how the (F/L)OSS programs are obtained by the people or what process they followed to select it

[1] For a systematic literature review see chapter 4.1

[2] Small and Medium Enterprises

has not been investigated thoroughly. Furthermore because (F/L)OSS is *libre*, it can't be procured. But knowledge (consultants), integration services and support subscriptions might be necessary to make the implementation a success and must be procured. This suggest that an intimate interaction between the IT-staff and the procurement department is necessary. But the literature does not consider this.

2.2 Issues in procurement practices

The procurement process as is widely understood contains at least the following steps: determining specification, selecting a vendor and contracting. Depending on the organization and kind of product that is procured this can involve more steps but this list forms the basics, the so called tactical purchasing part of procurement (Weele, 2005). These steps are implemented by procurement practices. Some of these procurement practices are at odds with the (F/L)OSS characteristics. Here three practices that are used in the procurement process are analyzed: requirements engineering practice, tender practice, total cost of ownership practice

2.2.1 Requirements engineering practice

In the determining specification step, the requirements of the system are determined. This can be done by requirements engineering. It is a common practice that the requirements[3] are vendor and technology neutral. Because the requirements are a formal functional specification of a narrative verbose need.

In theory (F/L)OSS always fulfills all requirements, because the customer receives the source code and can thus always adapt it to his or her needs. However in practice, software selection takes place by comparing existing alternatives. This comparison is made on the basis of the current states of the software programs ("as is"). This ignores the characteristic of (F/L)OSS, that a near future state of the software, might fulfill all your needs. This basically comes from the notion that software is seen as a product, something rock solid, instead of something stateless and fluid, like knowledge (Perens, 2005; Stallman, 1985).

Furthermore from news reports(Zwiekhorst, 2010) it can be concluded that sometimes the choice is reversed. Instead of looking for a solution that fulfills the requirements, people start with a proprietary product and then look for a (F/L)OSS *alternative*. So the logic of the choice changes from meeting the requirements to how identical the (F/L)OSS solution is to a proprietary product. This is for instance also seen in the action plan "Nederland Open in Verbinding" (NOiV[4]): "preference for open source software in the case of *equal* suitability"(NOiV, 2007). One might ask equal to what? A solution suits the requirements or it does not. This thinking is amplified by the recent trend of so called "reverse alignment" (Rosemann, Vessey, Weber, & Wyssusek, 2005). This is the practice where the organization is transformed to match the software, instead of the other way around.

[3] Requirement: "A function, constraint or other property that the system must provide to fill the needs of the system's intended user(s)" (Abbott, 1986)

[4] The name translates to "Netherlands Open in Connection". It is the name of the action plan and the agency monitoring (F/L)OSS use by the Dutch government

2.2.2 Tender practice

The second step in the procurement process is selecting vendors. In order to receive the best offer for a system that will fulfill the requirements, some organizations (notably governments[5]) make use of public tenders that stimulate competition (Seshadri, 2005; Ghosh et al., 2002). Tendering makes the assumption that there are vendors on the lookout for customers that want to buy their products. However for open source, this is not entirely[6] the case. This is because (F/L)OSS is not developed in companies, but in online projects. And these projects do not have the goal to make a profit. That means that the (F/L)OSS projects ("vendors") will not be looking for any "customers" (Waring & Maddocks, 2005; Morgan & Finnegan, 2007). This requires an active approach in finding vendors in stead of a passive one(Hoppenbrouwers, 2007; Eilander, 2008).

2.2.3 Total cost of ownership practice

To make a decision between the vendors, usually the Total Cost of Ownership (TCO) calculation is made (Sieverding, 2008; Varian & Shapiro, 2003). This is were organizations calculate all the costs of a product that is incurred during the product's life cycle (Ferrin & Plank, 2002). Suppliers of proprietary software products usually provide information on this life cycle. For instance the number of years the product is supported. Or how long an organization is allowed to use the product (license expiration dates in the end user agreement).

But (F/L)OSS projects supply the source code. This access to the source code provides the organization with the opportunity to maintain the software by itself. The organization really owns the software, instead of leasing it. So in theory, the lifetime of (F/L)OSS software is indefinite. Without an "end of life" date, TCO calculations become difficult. The question if maintaining the software by oneself is worth the effort, is not an operational question, but rather a strategic decision (Hahn, 2002; Lessig, 2002; Morgan & Finnegan, 2007; Glynn et al., 2005; Holck et al., 2005).

2.3 Research questions

Given all these inconsistencies from the previous chapter, it becomes clear that a difference exists in "procurement" of (F/L)OSS programs and proprietary software products (Hoppenbrouwers, 2007). The common procurement theories and practices seem unable to handle the subtle differences between the two paradigms (product vs knowledge). The effect of this can be read in the (daily) news about IT (Webwereld, 2008; ANP, 2009; Modine, 2009; Clarke, 2009; Dirven, 2009; Clarke, 2009). Also the (F/L)OSS literature does not appear to provide a clear answer on the "procurement" of (F/L)OSS yet either. So it can be concluded that there is a gap in the literature about how to deal with (F/L)OSS when "procuring" software. This is the first research question:

1 how do organizations deal with (F/L)OSS programs when procuring software for their operational processes?

But not all implementations of (F/L)OSS are failures! This is something that can be seen in the practical field. So why do some (F/L)OSS implementations succeed, while others fail? Clearly there

[5] Note that the tendering process might be demanded by law. This is for instance the case with government. Since (F/L)OSS programs are "gratis" official public tendering is not necessary for (F/L)OSS. This can be a large benefit, since tendering processes are expensive and take a lot of time (Oosterbaan, 2010).

[6] In the past decade large vendors like IBM, Sun and Google have been more involved with open source. These companies are looking for customers and offer open source software in addition to their proprietary products and services.

is something to learn from the practical field here. What have practitioners found out about the "procurement" of (F/L)OSS?:

2 What problems are found in these organizations when acquiring (F/L)OSS programs and have they found solutions for this?

Also not all the procurement theory should be disregarded. Could combining the practical solutions and existing theories fill this gap in the literature and help practitioners increase their success in implementing (F/L)OSS? Perhaps adjustments should be made in the "procurement process" to accommodate (F/L)OSS (Holck et al., 2005):

3 how can the procuring be improved?

The inconsistencies given in the previous section, the practical relevance and the gap in the literature provide ample reason to warrant research into the these research questions.

2.4 Overview of the report

This research consists of creating three sets of design principles. First theory-based design principles, which are created by a systematic literature review. Second practice-based design principles are extracted by a case study. And finally a third set of design principles is created by a synthesis of the theory- and practice-based design principles. These form the final design principles grounded in theory and practice.

All these components can be found in the results chapter (chapter 4) of the report, as shown in the list:

Systematic literature review: this is shown in chapter 4.1 : theory-based design principles

Case study: answers the 1st and 2nd research question. This is shown in chapter 4.2 : practice-based design principles

Synthesis: answers the 3rd research question. This is shown in chapter 4.3 : design principle synthesis

After the results chapter follows a discussion of the results, showing the contribution of the research (chapter 5). And the report is concluded with limitations and further research suggestions (chapter 6). But first the methodology is explained in the next chapter.

3 Methodology

The introduction and research question sections show that there is a gap in the literature between adoption and procurement of (F/L)OSS. This research tries to fill the gap by combining knowledge from theory and practice. A framework to achieve this is the research-design-development cycle Burg, Romme, Gilsing, and Reymen (2008) which is adopted in this thesis.

The research-design-development cycle framework used by Burg et al. (2008) provides a science-based design perspective that "links the scientific knowledge base" to "the pragmatic and creative work of practitioners" Burg et al. (2008). The link between scientific knowledge and practices are design principles which form the "boundary" objects between these two "worlds" (Romme & Endenburg, 2006). The design principles can "be used to create solutions that can subsequently tried out in practice" (Burg et al., 2008). The opposite path is also possible. Experimentation with solutions can serve to derive design principles (Plsek, Bibby, & Whitby, 2007). Figure 3.1 shows the framework and explains the two paths.

In this research two sets of design principles are extracted. One based on practical knowledge and one based on theoretical knowledge. Then these two sets are synthesized to create a final set of grounded design principles based on the theoretical and practical evidence. In figure 3.1 an overview is given of the research method which was adopted from De Jager (2009).

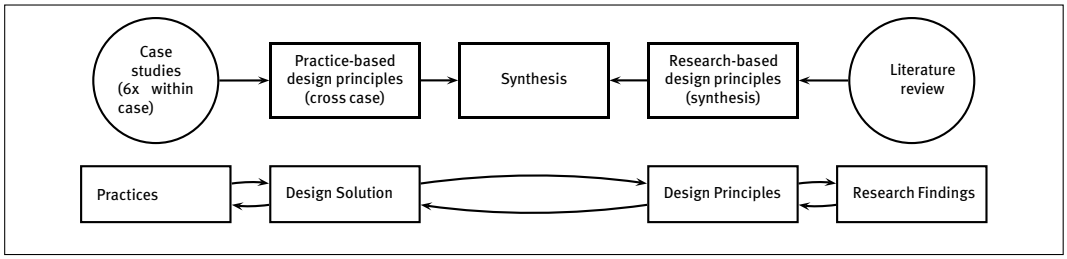

Figure 3.1: Schematic of the research design adopted from De Jager (2009) and Burg et al. (2008).

In order to extract the practice based design principles, a multiple case study has been executed. This case study consists of two parts. A within case study and cross case study. The within case study was used to corroborate the evidence of the interviewees, the documents and to build a structured summary of the case. This summary consists of two tables. One contains the process followed in the case to "procure" the (F/L)OSS (the process table). And the second table (the solution table) contains challenge, issues and solutions. These form a set (challenge-coping set), indicating a problem (challenge), the underlying reasons (issues) and the solution created in practice (coping). These two tables give a summary of each case that is in a format which allows the summaries to be compared in a cross case analysis.

The cross case analysis consisted of two steps. First the process followed in the cases was compared by combining the process tables from the cases built in the within case analysis (see section

4.2.2 for the result). Second all the challenge-coping sets were gathered. These sets were clustered and reduced by re-coding them to form a small set of recurring themes found within the data. These themes were then closely examined by reviewing the quotes in the interview transcripts and documents (see section 4.2.2 for the result). This made it possible to build design principles per theme (the result is table D.1).

Then a literature review was executed to extract the research based design principles from existing (incomplete) theory. These have been systematically induced by making use of qualitative meta-research synthesis as proposed by (Tranfield, Denyer, & Smart, 2003) and applied by (Burg et al., 2008). The method is a systematic way to generate evidence based knowledge from existing literature. The review focusses on the question how organizations "procure" (F/L)OSS.

The synthesis of the two sets of design principles can be done by following the techniques used by De Jager (2009) and Burg et al. (2008). This method combines the design principles found in both literature (the qualitative meta-research synthesis) and from practice (the case study) into one set of final design principles which are by then grounded in practice and theory.

Since the research cycle could not be completed within the limited time of this research it is stressed that only the generation of possible new design principles is executed and that these are not implemented and tested in the specific cases. This makes this research of highly *exploratory* nature. However some testing of the principles has been designed into research (see section 3.3.4).

The remainders of this section show how the literature review, the case analysis and the final design principle synthesis were executed[1]. But first the scope of the research is determined.

3.1 Research scope and goal

The research questions were broadly defined. This would make the research extensive. In order to make the research feasible, the scope was reduced. First by narrowing down the type of organization to government organizations. Second by stating explicitly what is being investigated (unit of analysis). And finally the goal of the research was formulated clearly, based on the reduced scope and the original research questions.

3.1.1 Government organizational context

The Dutch government was chosen as the context for the research from a practical point of view because the Dutch government is working on "using open source" (Willems & Klip, 2009). This provided a pool of possible cases and thus an opportunity to investigate the procurement. Furthermore the "programma bureau Nederland Open in Verbinding (NOiV)", a Dutch government agency, is tasked to keep track[2] of the (F/L)OSS use. Therefore the public records of the NOiV were used to find out which parts of the Dutch government are currently working on (F/L)OSS procurement or have done this in the recent history.

Choosing a government setting, restricts the element of organizational culture that might be a factor in the research. The research has been executed on an operational level. The focus was on the IT-staff. And their culture is relatively similar across organizations, because they have had the same education, use the same tools and some are part of the hacker culture (Nichols & Twidale, 2003;

[1] The literature review followed the case analysis during the research. This made sure the coding of the case analysis was not hindered by knowledge and constructs from theory.

[2] They also provide advice about (F/L)OSS to any and all layers of the Dutch government.

Krishnamurthy, 2006; Raymond, 1999)). Therefore the research is believed to be generalizable to other organizations under similar conditions (also see the limitations chapter 6).

3.1.2 Unit of analysis

The research question is how do government organizations "deal with (F/L)OSS programs". This offers a rather vague unit of analysis and to make it more clear, a closer look was taken at the term "procurement" (Weele, 2005). From figure 3.2 it becomes clear that procurement is a container term

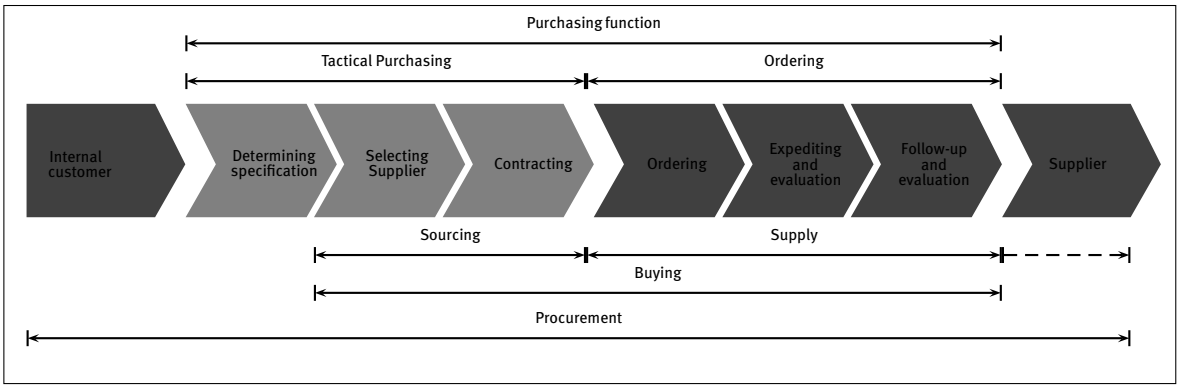

Figure 3.2: Purchasing process model and some related concepts. Adopted from van Weele 2005

for several functions. The research question was to find out about how organizations "deal" with (F/L)OSS programs. Thus narrowing procurement down to the tactical purchasing[3] frame seems a just strategy because this entails determining specifications, selecting vendors and contracting. So it is believed this frame will capture reasons for choosing (F/L)OSS as well as any thoughts or discussions about the implementation in the organization and all the challenges that come with it. Therefore the following description of the unit of analysis is given.

> The unit of analysis will be the tactical purchasing process of software (Weele, 2005). This is believed to be a decision process where a decision is made to use a particular (F/L)OSS program. This can be any software program for any use but it has to have at least an Open Source Software (OSS) qualification (so have an OSI certified license (Coar, 2006)). Furthermore the decision has to have been made consciously. A group of people need to have been involved in the decision making process. That way it is believed that the choice is made consciously and the process can be made visible.

3.1.3 Goal

After limiting the scope and defining the unit of analysis, it was possible to explicitly state the goals of the research. This research had the goal to: **generate new knowledge about (F/L)OSS procurement by government organizations for the academic and practical field.** It has generated exploratory insight of how government organizations currently execute the tactical purchasing of (F/L)OSS programs.

Next to this goal of the research, there is also a goal *in* the research (Burg et al., 2008). This goal in the research is: **to develop a set design principles that will help government organizations design their tactical purchasing policies to include (F/L)OSS.**

[3] Also the ordering side of procurement is not interesting for this research, since logistics are not much of an issue for software. Downloading, CDROMS, DVDs and central servers make distributing software rather easy, compared to, for instance, making sure machine parts arrive just in time to a production plant situated on another continent.

Initial	Expanded
Procurement of (F/L)OSS	Entrepreneurial behavior
Adoption of (F/L)OSS	Top management
Government procurement of (F/L)OSS	Product champion
Factors using (F/L)OSS	IT architectural models
Inhibitors using (F/L)OSS	
Software acquisition	
Government tender	
Decision making	

Table 3.1: The keywords used in the literature review.

3.2 Systematic literature review and synthesis

A systematic literature review was performed in order to extract research based design principles from literature (Pawson, Greenhalgh, Harvey, & Walshe, 2004; Burg et al., 2008; Denyer, Tranfield, & Aken, 2008). First a scope was determined based on the research questions. Then a set of key words has been generated. These were based on the scope and also by making use of synonyms of the key words. Then the ABI/inform, (F/L)OSSpapers.org, mit.opensource.org databases and the Google scholar search service were used to find relevant articles.

3.2.1 Scope and key-words

It has to be understood that the literature review was a continuous effort during the entire time of the research. From this predisposed knowledge the initial set of keywords were determined (as was done in Denyer et al. (2008)). Please also bear in mind that the review is iterative (Pawson et al., 2004). Therefore the set of keywords actually evolves (edited and expanded) during the research. To demonstrate this evolution, table 3.1 is included where the initial and final set of keywords are displayed.

Articles about the (F/L)OSS development process were not considered. This topic is irrelevant for this research, but is well documented and thus creates a high noise level when searching within (F/L)OSS literature.

By reading the abstracts of the articles a further selection has been made. Also the snowball[4] method has been used until saturation occurred and the program theories became apparent (Pawson et al., 2004). Note that the specific literature field has not been stated. This was done in order to allow literature from different fields of research to contribute to the review as was suggested by Romme and Endenburg (2006). This caused an article from the software development field about reverse alignment (Rosemann et al., 2005) to be included in the review.

3.2.2 Extraction of theory-based design principles

Pawson et al. (2004) explains a realist synthesis. This literature review strategy has the goal to gather actions/solution from a research programme in a systematic way. This method can therefore be used to identify design principles in a structured way. Plsek et al. (2007) describes a method to generate research based design principles from documentation. This method is adopted here and combined with the method used by Burg et al. (2008). Together with the CIMO logic introduced by Denyer et al. (2008) a set of research based design principles has been created.

[4] This method uses the references of a relevant article to find other relevant articles.

3.3 Case study

The practice based design principles have been generated by a multiple case study (Eisenhardt, 1989). The case study has followed the steps as set forth in the article by Eisenhardt (1989). These steps are case selection, crafting instruments and protocols, analyzing data, theory building and quality control (Eisenhardt, 1989). The next sections describe each of these steps in detail.

3.3.1 Case selection

In order to find out relevant factors to determine the case selection a short case exploration has been performed before the research began. This also conveniently assessed the feasibility of the research because it made clear that there was enough data in the form of cases to be found in the field. For this inquiry, three interviews were held with practitioners that have experience in the field and access to case data of multiple cases.

From the inquiry it became apparent that the type of software might be an important factor (also stated by (Cassell, 2008)). There are three classes. Off the shelf (OTS) software. Software that needs skilled personnel for configuring and installing (infrastructure software). And software which is designed by the organization itself.

This research is about how to "procure", that is why only (F/L)OSS programs that can be *"procured"* are looked at and not the programs which are *developed* by the government organizations. Because no procurement takes place with software that is created by the user itself. However if the case has a make or buy decision, than this case will be considered, since that can be a strategic choice.

From the literature it is also expected that the users will be different depending on the type of software (Hippel, 2005; Jullien & France, 2009; Grand, Krogh, Leonard, & Swap, 2004). And there is evidence that the maturity of (F/L)OSS programs plays a role (Munoz-Cornejo et al., 2008).

Furthermore it is interesting to make a distinction between operational factors and strategic influences. Because in the short investigation it was found that the decisions are sometimes made on the basis of rather operational demands than on strategic goals.

Why a case study? A case study is performed because the research question is a "how" question, where the "boundaries between phenomenon and context are not clearly evident" (Yin, 1981). Furthermore the research questions are about a contemporary phenomenon so the case study is the best choice for the research (Yin, 1981). It was also uncertain if a "procurement process" would be found in the field of practice. This research tries to *explore* a phenomenon namely to gain insight in the tactical purchasing of government organizations. That's why an explorative and multiple case study is done (Eisenhardt, 1989; Burg et al., 2008; Benbaset, Glodstein, & Mead, 1987) also because the study is not about a single unique case but a phenomenon(Yin, 1984). According to the process study design literature the situation of the present research is such (few cases and exploratory) that A. Van de Ven (2007) recommends using a comparative case study design following Yin. And another reason for the application of a multiple case study is that more than one case is necessary in order to generate sufficient data for triangulation and thus valid evidence for the design principles (Romme & Endenburg, 2006). Also per case more than one source is used (see also section 3.3.4) (Plsek et al., 2007; Yin, 1984) to ensure rich cases. The methods introduced by Plsek et al. (2007) are used to extract practice based design principles from the case study data in the analysis.

For these reasons the case selection consists of three cases of a mature COTS type and three cases with a mature infrastructure/development type of software. The cases in which the organization has made its own software are ignored. Also to contrast the operational factors with the strategic factors, the cases are chosen in big and small organizations and with traditional high strategic foresight (department of defense) and low strategic interest (municipalities).

The interviews were held at the Dutch government (called "Het Rijk") and Dutch municipalities see list in table 3.2. Furthermore, other documents were used, some provided by the case members themselves and others gathered from public sources. One of those source was the NOiV website,

Name	Tag	Type	Description	Hired knowledge	# users	Evidence
Dutch Patent Office	DPO	Infra	Open desktop	Yes	130	NOiV case study, business case, 3 interviews
DoD IVENT	DoD	Infra	Project PODVIS and PORSELIJN	Internal R&D	60000	Project initiation document, 3 interviews
Case Z	Case Z	Infra	...	...	...	3 interviews, documents regarding explanation of policy
Tax Office	DTO	OTS	Project pdfCreator	No	30000	1 interview with 3 persons, documents
Grootegast Municipality	Grootegast	OTS	Open Office	Yes	80	business case, 3 interviews
Voerendaal Municipality	Voerendaal	OTS	Open Office	Yes	80	NOiV case study, presentation, 3 interviews

Table 3.2: Overview of the cases.

which provides additional case descriptions about some of the investigated cases and general manuals for the use of (F/L)OSS in a government setting. All these documents made sure that rich cases were created (see also section 3.3.4).

In order to gain access to the case study sites the research was supported by the Gendo company. This is an IT management consulting firm, that advises organizations on IT-strategy. The company also executes and manages ICT projects. This company is well known in the Dutch open source community, since the company believes in social responsibility and this is demonstrated by investing a considerable amount of time in the promotion of (F/L)OSS. For instance one of the company's founders, Arjen Kamphuis, contributed to the motion that proposed the use of open source and open standards in the government (NOiV, 2007). By leveraging this network a lot of access to cases, the government and the Dutch open source community was obtained.

3.3.2 Instruments and protocols

For the collection of the evidence a protocol was developed as advised by Eisenhardt (1989) The questions were primarily based on the research questions directly and used so called probes (Miles & Huberman, 1994). These probes were based on a previous literature review (Verheesen, 2010). The protocol makes sure a consistent routine is followed in the interview. That way, relevant factors and case characteristics could be identified. For the sake of conciseness, the protocol can be found in B.

Four methods to elicit design principles are explained by Plsek et al. (2007). In this research three of them are used. These are the use of documents, storytelling and hypothetical scenarios. The fourth, making change leaders discuss the process in terms of design rules, is not used. This is because the disadvantages outweigh the benefits (Plsek et al., 2007). It would make the interviews longer and much more complex and thus the total study less feasible.

To implement the storytelling and hypothetical scenarios, the protocol (i.e. the questions asked in the interview) allowed a particular order. First the interviewee was asked to freely relate what he thought were relevant events in the process. Then questions were asked to clarify the process. After this was clear, the questions became more leading and tried to make the interviewee think about the strategic issues at stake. By letting the interviewee build a hypothetical scenario as described by Plsek et al. (2007) more insight was gained to the reasons why the choices where made.

After the data collection, the coding started. To prevent researcher bias, the planned open coding was changed to a limited set of codes with a general meaning instead. Because using open coding it is tempting to draw premature conclusions whilst coding. The predefined codes were more objective and closely related to the elements of the design principles, which made the consecutive cross case analysis more straight forward. The codes used were: strains, issues, how_coped,

how_resolved and process_step. For each fragment of interview the question was asked: is the interviewer describing a step in the process, a problem, reasoning about something or a solution? This is much more objective than directly applying open coding.

3.3.3 Extraction of practice-based design principles

After the coding the analysis began. This research consists of two analysis. First a within case analysis was made in order to reduce and refine the amount of data and then a cross case analyses in order to generate the design principles. In this section these two analysis are described.

For each of the six cases a within-case analysis was made. This provided a structured and detailed description of each case. In the analysis all the quotes belonging to the strains, issues, coping and resolving codes of the three interviewees were gathered in these five groups. By focusing on one group (for example reading all the text classified as perceived strains) it is easier to find common themes across interviewees. These themes were then coded by using the open coding process. Analyzing this way answered the questions: Which fragments describe the same strains and give reason for a real strain? What did they do to cope? What are the causal links between these problems and solutions found in the narrative? What is the reasoning behind the situation and the solutions? This provided a reduction in data while maintaining a detailed and structured description of each case.

This way the combined 130 (average) strain-coping sets of the interviewees were reduced to 5 or 6 strain-coping sets describing the main problems of a case. These are summarized in the tables in this report. This summary and the complete analysis was then given to the interviewees for them to check. This was meant to find out if the main strain-coping sets were in fact real issues and to prevent bias (reading into the case). It also provided an opportunity for the interviewees to comment on the findings and to suggest or clarify points. This input was then used to adjust the within-case analysis.

The same process was followed for the process_step code. The analysis method almost immediately formed a chronological order inside the spreadsheet. And thus a summary of the steps taken in the case could be made quickly.

The results of the within-case were the input for the cross case analysis. This analysis was used to derive the practice based design principles. By reviewing all the strain-coping sets of all the cases, one can notice similarities. A table was created, putting the cases in a column and the strain-coping set in the corresponding row. Then by shifting and sorting similar strain-coping sets can be found. Eventually these were placed their own columns. These columns are called themes.

In total 6 distinct recurring themes were found. These themes were then examined upclose, by reviewing the interview transcripts in depth. This was done theme by theme and resulted in design principles per theme. Because all the narrative is already in strain-coping sets and now ordered cross case by common theme, it is straight forward to mould this information into CIMO-logic form. This form is used as a display for the design principles. This resulted into 8 main design principles, usually with more than one intervention.

3.3.4 Quality control

In order to provide a thorough case study, attention was paid to the chain of evidence, triangulation, reliability, validity, and rival explanations. In this section the use of methods to control these factors are explained.

Chain of evidence The chain of evidence is important in this kind of research, to ensure that the formulation of the design principle can be traced back to elicited answers (quotes) in the interviews and documents (Yin, 1984), thereby showing the proof for the answers to the research questions. Eisenhardt (1989) advises to create protocols and tools for the collection of the interview evidence. As was already shown these protocols and tools have been created in order to create a clear chain of evidence. The chain of evidence was implemented with the help of computer programmes during the research. Consecuentlyly during the creation of this report, much effort went into making clear what the foundation of the claims are and results that are made.

Triangulation The primary source of information in this research will be the stories told by the interviewees. These people are carefully selected and stem from different (hierarchical) parts of the investigated organization. Per site, a policy maker, project leader and project member have been interviewed.

Yin (1984) states different sources of evidence that can be used in a case study. In this case study three sources of evidence are used. Case-related documents used to corroborated the stories told in the interviews, the interviews themselves and documents of public sources about the case are used. Where possible contact will also be made with companies involved to get a complete picture of the process that was carried out. By combining the information from these sources triangulation can be performed and thus rich cases can be created.

Reliability, validity, and rival explanations During the within-case analysis an indication for interrater reliability was determined on the set of objective codes. This was done by having three people code one interview. This resulted[5] in a moderate interrater reliability (Kappa = 0,49). In order to give the developed design principles more validity, the final design principles have been tested by an expert-opinion model. This works by having experts give their insights on the newly developed set of principles in a design principle review. Thereby creating more knowledge about the indications and contra-indications of the theory. This was done by presenting the interviewees with the concept report and asking them to express their opinion about the derived design principles.

Sometimes in the cases and in the literature, contradicting or alternative explanation were found. Where applicable these explanations are mentioned in the report and their implications were determined. Also in appendix F provides room for these rival explanations.

Together with the mentioned triangulation, the use of rich cases, showing rival explanations, chain of evidence procedures in the case study, determining interrater reliability and executing a design principle review it is hoped insight is given in the validity and reliability of the output.

3.4 Design principle synthesis

The literature review provides the design principles from theory. The case study gives design principles from practice and the final set of design principles is created by synthesizing these two sets into one. This can be done by following the CIMO logic. In this section first the CIMO logic is explained further and then second it is explained how this logic was used to synthesize the final principles.

[5] This number was calculated by cat.ucsur.pitt.edu an (F/L)OSS based webapplication for qualitative data analysis from the University of Pittsburgh. See appendix G

3.4.1 CIMO logic

CIMO logic is a way to formulate design principles. Design principles have the goal to explain why (mechanism) acting in a particular way (an intervention) in a certain environment (the context) will cause an outcome. The CIMO logic framework thus provides a systematic and concise way of formulating design principles. CIMO-logic stands for problematic context, intervention, mechanism and outcome. The follow list has been adopted from (Brouwer, Brekelmans, Nieuwenhuis, & Simons, 2010).

Context Problematic context constitutes surrounding (external and internal environment) factors and the nature of the human actors that influence behavioral change (Denyer et al., 2008).

Interventions Interventions are purposeful measures (products, processes or activities) that are formulated by the designer or design team in order to solve a design problem or need (Denyer et al., 2008; Midgley, 2000).

Mechanisms Mechanisms indicate why the intervention produces a certain outcome. The mechanism is triggered by the intervention in the context (Denyer et al., 2008) and is an account of the cognitive processes (reasoning) actors use to choose their response to the intervention and their ability (resources) to put the intervention into practice (Pawson & Tilley, 1997; Van Aken, 2005). Underlying conditions for the actions of actors are for instance motivation, team culture, financial incentives and policy (Pawson & Tilley, 1997).

Outcome Outcomes are the result of the interventions. CIMO-logic determines that a design principle has the following structure: "in this class of problematic contexts, use this intervention type to invoke these generative mechanism(s), to deliver these outcome(s)" (Denyer et al., 2008, page 395).

3.4.2 Synthesizing of final design principles

Because the design principles have been formulated in CIMO logic, they can be compared. This was done by first looking at the context. Gathering all the design principles with the same context. Then per context, the corresponding interventions were searched for overlap. Overlap could provide a clearer statement of the precise intervention, or could just mention the same. The mechanisms were also compared. In nested interventions, the corresponding mechanism should align by giving a more explicit description of the mechanism that is at work.

After this all the outcomes of all the principles were searched. Sometimes the outcomes of a design principle can create a different context. That could couple the design principles. In most cases however the outcomes could be either added or were describing the same outcome.

4 Results

This chapter consists of three sections. First section shows how the theory-based design principles were developed. The second section presents the extraction of the practice-based design principles from the case studies. And the final section demonstrates the synthesis of these two sets of design principles into a final set of principles, then grounded in both literature and practice.

4.1 Theory-based design principles

With a systematic literature review and synthesis the theory-based design principles are developed here. The literature review covers three areas, adoption, IT architecture and procurement. First adoption from the (F/L)OSS literature was studied, because it focuses mainly on antecedents that explain why adoption takes place. The same line of argument can thus be used to search for methods that cause adoption. During the case study research, it became apparent that also IT architecture was relevant. Because through architecture it is possible to steer acquisition of software. Therefore this literature was also included. And finally, procurement topic was included because in practice software products are procured by a procurement process. Understanding which steps there are in the process and what parts conflict with (F/L)OSS might provide solution directions. Therefore this topic was also included.

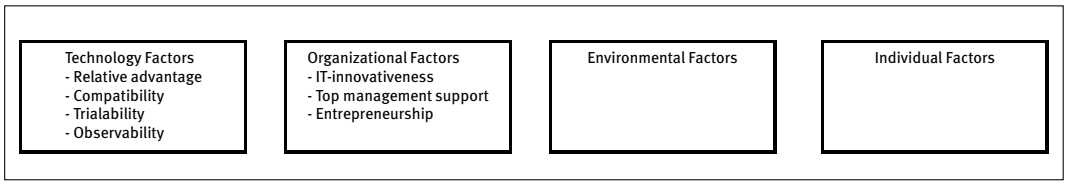

Figure 4.1: Overview of the relevant adoption factors.

4.1.1 Adoption factors

In this review four adoption factors are investigated (technological-, organizational-, environmental- and individual factors). Widely used throughout the (F/L)OSS literature is the adoption model of Depietro, Wiarda, and Fleischer (1990). This basic model consists of the first three factors (technological, organizational and environmental factors). That model was used in (Dedrick & West, 2004). Glynn et al. (2005) expanded this model in (F/L)OSS context with individual factors. In this review the original model combined with the expansions is used to search for design principles (see figure 4.1 for a display of the four factors that are investigated).

Technology factors

The technology factors can be split up in five sub-factors by using the framework of Rogers (1976). These sub-factors are: relative advantage, complexity, compatibility, trialability and observability. Four of these factors are explained in the following sections. Fichman and Kemerer (1993) combines the five sub-factors from Rogers (1976), with four economic factors from the standards adoption theory (prior technology drag, investment irreversibility, sponsorship, expectations). Only the two most relevant economic factors (prior technology drag, investment irreversibility) are used here and combined with the compatibility sub-factor. Because the rest of the economic factors has even more overlap with the sub-factors from Rogers (1976). In the next sections, the four sub-factors of Rogers (1976) (relative advantage, compatibility, trialability and observability) are explained.

Relative advantage The most important relative advantage factor found in the literature was cost (Valimaki et al., 2005; Koh, 2009; Hwang, 2005; Ozel et al., 2007; Glynn et al., 2005; Evans & Reddy, 2003; Ghosh et al., 2002). This is spread over two arguments. Software licenses and switching costs.

Since (F/L)OSS is "gratis" and so are upgrades there are no ongoing costs to stay with the latest (F/L)OSS programs unlike proprietary software (Dedrick & West, 2004; Miralles, Sieber, & Valor, 2005). If the licensing model is based on "per seat" licences than (F/L)OSS is an attractive alternative (Masrek, Jamaludin, & Hashim, 2009)

Switching cost in the context of retraining can vary, depending on the present skill of the IT-staff (Dedrick & West, 2004; Varian & Shapiro, 2003). Furthermore switching cost can contain investments in customized software, if close standards have been used the conversion costs include de-integrating software. The last argument requires explanation. These are the costs incurred when a product that provides two different functions must be replaced. Usually the "unused" functionality is actually found out to be used. This means the product cannot be replaced(Centra, 2006).

Ghosh (2010) argues that these costs should be added to the costs of a proprietary system when comparing it with the cost of a (F/L)OSS system. This is because much of these costs will not occur, when the solution is based on (F/L)OSS. So these long term exit costs belong to the price of the proprietary solution that must be paid today(Varian & Shapiro, 2003).

Theory-based design principle 1

c In the context of formulating arguments to support adoption of (F/L)OSS,

o determining the cost for a (radical) (F/L)OSS implementation can be done,

i by calculating the switching cost what it would cost to implement the proprietary software now and change to (F/L)OSS later,

m because these costs belong to the proprietary systems since they would not be incurred when adopting for (F/L)OSS now.

Compatibility factors Two compatibility factors are found to be the most important. First, technical compatibility with legacy software. And second, compatibility with existing knowledge.

One of the greatest adoption inhibitors is technical incompatibility with existing software. This is called prior technology drag factor in the standards model and a more popular term is "vendor lock-in"(West & Derick, 2006). This is caused by legacy systems that are mission critical yet cannot be ported to, or connected with, a new ((F/L)OSS) platform and/or programs (Miralles et al., 2005; Hauge et al., 2008). And the prior technology provides network benefits because of large and mature installed base (Fichman & Kemerer, 1993). There are two solutions for this.

First one is replace the complete part of legacy software with a (F/L)OSS alternative if such an alternative exists. This can have the disadvantage of sunken costs (Nagy et al., 2010; Kauffman & Li, 2003), because changing technology requires irreversible investments in training and products (Fichman & Kemerer, 1993). Unlike proprietary products, (F/L)OSS has the unique characteristics (freedom to: use, alter, (re)distribute the software) that makes vendor lock-in impossible ().

Second is the encapsulation of the proprietary software. This can be done by installing middleware that transforms the proprietary output to an open format (Nagy et al., 2010). Or by using virtualization software. This software can be perceived as a bubble isolating the proprietary software and this bubble than resides in an all (F/L)OSS environment.

Theory-based design principle 2

c In the context of technical compatibility issues with legacy technology,

o to reduce lock-in,

i by replacing the legacy systems by installing (F/L)OSS (alternatives) as much as possible,

m because the characteristics of (F/L)OSS make vendor lock-in impossible.

i by encapsulating the parts that cannot be replaced, by using middleware and virtualisation techniques,

m because these will make it possible to limit the amount of the legacy software and thus increasing the opportunity to use (F/L)OSS.

Compatibility with existing knowledge means end-users can easily adapt to the new technology by using their existing knowledge. Training can be used to overcome a minor knowledge gap. This increases the user friendliness and usability of the software (perceived ease of use, self efficacy in using the technology (Y. Lee, Kozar, & Larsen, n.d.)). End-users should receive training as soon as the new tools are fully functional in place (Zuliani, 2004; K. Ven, Huysmans, & Verelst, 2007; Zuliani & Succi, n.d.). "A delay between training and productive work with the new tools will diminish the effect of the training itself" (Zuliani, 2004). However in the case of a radical difference in required knowledge, more management is necessary.

Depending on the existing knowledge and skills of the IT-staff, (F/L)OSS can be radical technology. For instance in literature it is noticed that the transition from UNIX to Linux is relatively easy (Morgan & Finnegan, 2007). But from Windows to Linux or Linux to Windows is hard (Dedrick & West, 2004; Glynn et al., 2005; Ajila & Wu, 2007; Varian & Shapiro, 2003). This is because most Linux distributions contain the GNU system. A component meant as a free alternative to UNIX (GNU stands for GNU is Not Unix) (Stallman, 1985). So Linux and UNIX are similar, while Windows is completely different. Therefore a change towards (F/L)OSS use can sometimes be radical (Miralles et al., 2005). In that case the compatibility with existing IT-skills can be an inhibiting factor of (F/L)OSS adoption (Nagy et al., 2010; Glynn et al., 2005; Dedrick & West, 2004).

Whenever making this radical change (Miralles et al., 2005) some form of knowledge management must come into play. Either the knowledge comes from outside (Nagy et al., 2010) or the knowledge is fostered within the organization. Both methods require knowledge management to diffuse the knowledge internally, because of the stickiness of knowledge and to prevent the not invented here syndrome (Szulanski, 1996; Cohen, Keller, & Streeter, 1979).

To gain access quickly to (F/L)OSS knowledge it can be procured externally (Valimaki et al., 2005; Nagy et al., 2010). External hire can increase the knowledge by cross-training and spreading awareness within the organization (Nagy et al., 2010).

A more structured way, which therefore requires more time is institutional skill building (Woods & Guliani, 2005). Carbone and Architect-Nortel (2007) call this the (F/L)OSS adoption learning ladder. This model describes how an organization can go from only using (F/L)OSS to producing

(F/L)OSS. Because the source code is available this provides the opportunity to expand knowledge in the IT-staff (Woods & Guliani, 2005). And following the model, along the way an organization becomes more proficient in using and altering it for its purpose. Farbey and Finkelstein (2001) even goes as far as proposing a learning ladder in order to create the "competitive advantage of the acquisition of software". The proposed model is based on network sourcing which resembles the (F/L)OSS development model (Kogut & Metiu, 2001). So the final state of an organization is being able to adapt the organizational business processes, have the knowledge capabilities and skills to translate it into software to automate the task and thereby satisfy the strategic needs. This requires capturing IT-staff skill and taking charge of the IT-infrastructure by documenting and systematically preserving the (F/L)OSS skills and knowledge (Woods & Guliani, 2005).

Theory-based design principle 3

c In the context of technology factors influencing the adoption of (F/L)OSS,

o create compatible knowledge and match the level of skill required for the use of (F/L)OSS,

i by training employees in the technology

m because training will increase the usability of the (F/L)OSS software and thus the usefulness.

Trialability Trialability means that an innovation can be experimented with without much effort and expense (Dedrick & West, 2004; Fichman & Kemerer, 1993). On this factor (F/L)OSS scores points since it can be freely downloaded, shared, installed and tested. This is key in determining if the solution fulfills requirements and offers the possibility to let end-users experience the innovation. That way ensures that the right solution is chosen and that the end-users get a feeling of being listened to, and thus they will properly support the new technology.

Observability The final influential factor is observability. "The results and benefits of the innovation's use should be easily observable and communicated to others" (Fichman & Kemerer, 1993). The more visible the results of technology the more likely the innovation will be quickly adopted (Tornatzky & Klein, 1982). Training influences the formation of user perceptions on: ease of use, compatibility, observability, and trialability (Venkatesh, 1999; Xia & Lee, 2000).

Training "moulds" these user perceptions. In the beginning these are vague and tentative in nature, because they are based on limited knowledge and experience (Xia & Lee, 2000). When users are able to have a direct-use experience they develop a better understanding of the strengths and weaknesses of the IT innovation. This enables the users to change their initial expectations to more realistic assessments (Xia & Lee, 2000).

Theory-based design principle 4

c in the context of technology factors influencing adopting (F/L)OSS

o creating support from end-user can be achieved

i by increasing trialability, letting end-user play with the solution

m because having end-users experience the solution will enable the possibility to test if the solution fulfills the requirements and will give the end-user a feeling that their wishes are being heard.

i by increasing observability, show (training, pilots) employees that the solution creates benefits for them by communicating experiences and results

m because the more visible the results of technology the more likely the innovation will be quickly adopted.

Organizational factors

IT innovativeness Another factor is IT innovativeness. This is a strategic question. How much does the organization want to innovate? This determines which kind of cues will trigger the decision to adopt (Dedrick & West, 2004). One way to answer this, is by having top management issue a statement of strategic intent (Hamel & Prahalad, 2005). This is an ultimate goal of an organization for instance "beat benz" in the case of a Japanese auto manufacturer. Strategic intent captures the essence of winning and sets a target for personal effort and commitment. Strategic intent is clear about ends, but flexible in means. It implies a sizable stretch for an organization. Because current capabilities and resources will not suffice, the organization is forced to be more inventive and make the most of limited resources (Hamel & Prahalad, 2005).

Theory-based design principle 5

c In the context of organizational factors influencing the adoption of (F/L)OSS,

o obtaining personal effort, employee commitment and triggering adoption, is done by

i by issuing strategic intent (a clear in goal) but being flexible in means to reach this goal,

m because current resources and capabilities will not suffice and thus the organization is forced to be more inventive and make the most of the limit resources.

Top management support Top management support is critical for radical and high-risk initiatives (Glynn et al., 2005). In some cases (F/L)OSS will fall in that category. For instance Tushman and Anderson (1986) notes that a transition to (F/L)OSS can be discontinuous, depending on the skill set of the IT staff. Moreover the amount of knowledge that organizational members must acquire to adopt the technology is an index for the radicalness of an innovation (Rogers, 1995).

In order to help the adoption, top management support can be given by supplying resources. These could already exist. So-called Slack resources are a set of resources perceived to be in excess (Joo & Kim, 2004). But top management support also has the power to distribute the necessary resources such as time, space, equipment and people, which will make adoption easier. And top management has the legitimacy that can create clear common objectives toward technological advances in the organization by communicating, supporting and sharing, vision and goals (Masrek et al., 2009).

Resources are usually mentioned in the literature in two dimensions, human and financial (Masrek et al., 2009; Glynn et al., 2005). When for example an organization has slack in IT staff (human resources) and they have the compatible skills, than "opting for a 'gratis' (F/L)OSS operating system without support subscription makes sense" (West & Derick, 2006).

Financial (slack) resources can also be used to acquire resources that aid in implementation, such as securing the services of managerial or technical talent from a consulting firm (Masrek et al., 2009).

So (slack) resources can offer room (time, money) for experimentation. Resources also allow for more risk to be taken, because the "cushion of slack assets will mitigate the blow of a failure should it occur" (Masrek et al., 2009).

Theory-based design principle 6

c In the context of radical, high-risk initiatives and absence of slack resources,

o obtaining necessary resources such as time, space, equipment, people and communicating, supporting and sharing, vision and goals is done:

i by creating top management support,

m because these resources are vital to the success of such an initiative and only top management can provide these resource because they have the legitimacy and power to distribute them.

Entrepreneurship Individual entrepreneurialism is associated in the literature with the freedom to conduct experiments (Stopford & Baden-Fuller, 1994). Usually the person displaying this behavior is called an entrepreneur or product champion. The literature suggests that there should be such an enthusiastic person, with room for experimenting with (F/L)OSS (Cohen et al., 1979; Zuliani, 2004; Woods & Guliani, 2005; Fichman & Kemerer, 1993).

Top management support can help foster this innovative behavior among employees in the organization (Cassell, 2008). Individual entrepreneurialism happens in stage 1 of intra-organizational entrepreneurship. This behavior can continue to grow into renewal when fostered (Stopford & Baden–Fuller, 1994).

Fostering can be done by making sure the organization provides two circumstances. First a precise definition of a strategic goal must be given to understand the problem and/or opportunity. And second, there must be a high perceived capability for solutions (Stopford & Baden-Fuller, 1994). Only then internal action will follow.

This action can either lead to turmoil or renewal, depending on the perceived nature (in terms of scope and urgency) of the required action. Because a sense of urgency is necessary to change turmoil into renewal (Stopford & Baden-Fuller, 1994; Hamel & Prahalad, 2005). For instance a lack of urgency combined with (slack) resources, can reduce discipline (i.e. the focus on the strategic intent message (Hamel & Prahalad, 2005)) and lead to investments in pet projects with limited economic value (Dedrick & West, 2004).

Theory-based design principle 7

c In the context of organizational factors influencing the adoption of (F/L)OSS,

o individual entrepreneurialism should be fostered,

i by stating a precise definition of a (urgent) strategic goal

m because this will lead to an understanding of the challenge and helps identifying an opportunity

i by organizing a high perceived capability for solutions,

m because knowing what the team is capable of doing will also identify what the possibilities for a solution are and what capabilities must be acquired in order to implement the solution.

Environmental factors

One of the most found influential environmental factors is marketing of (F/L)OSS. This has appeared in three ways. First, one has to simply know that (F/L)OSS exists as an alternative (Nagy et al., 2010). Second the image that (F/L)OSS has, and thirdly because of network externality.

It is very possible that people within an organization are unaware of the fact that something like open source software exists, let alone the specific names of (F/L)OSS programs (Nagy et al., 2010; Morgan & Finnegan, 2007; Krishnamurthy, 2003). This is could be because “marketing” of the program could have the least priority within the (F/L)OSS projects (Raymond, 1999; Krishnamurthy, 2003). The solution is to actively “monitor open source archives” like sourceforge and freshmeat.net, “training staff members in installing, using and customizing (F/L)OSS”, select new personnel on the

basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines “to keep up with the latest trends and events in this area” (Nagy et al., 2010).

There are a lot of misconceptions about (F/L)OSS which give it a bad reputation. Next to the regular misconceptions: legal minefield, no support, savings are an illusion, it’s socialism, it destroys intellectual property, it only moves vendor dependence (Souabi, 2007; Miller, 2002; Wheatley, 2004), most misconceptions are about the “maturity” of the software (Souabi, 2007). This is solved by applying software maturity models on open source (Aberdour, 2007; Zhao & Elbaum, 2003; Ajila & Wu, 2007). Furthermore it appears that larger organizations prefer the vendor support from ’big IT’ (IBM, HP, SUN etc.) “the fact that HP is committed to Linux is comforting” (West & Derick, 2006). This provides decision makers with a feeling that (F/L)OSS is legitimate. This also has to do with individual factors, because “CIO’s do not want to be associated with having chosen the ’losing platform”’ (Miralles et al., 2005).

Network externalities in the form of presence of (F/L)OSS knowledge are important. That way organizations that lack the knowledge required to implement and use (F/L)OSS, can procure the knowledge externally (Nagy et al., 2010). But there appears to be a lack of (F/L)OSS skills in the industry (Koh, 2009; Giera, 2004). A possible explanation might be that there is also a lack of knowledge about organizations offering support for (F/L)OSS (Dedrick & West, 2004; Valimaki et al., 2005; Nagy et al., 2010). Next to ’big IT’ most (F/L)OSS support is given by local companies only known in their area to support (F/L)OSS (Valimaki et al., 2005). However, with more (F/L)OSS adoption a situation can arise in which “a network of interested parties with complementary capabilities can form an ecosystem to offer a professional product and service in an agile, bazaar-friendly manner” (Fitzgerald, 2006). This looks like the network sourcing also mentioned in the compatibility factor.

Theory-based design principle 8

c In the context of external environmental factors influencing (F/L)OSS adoption

o creating more awareness about (F/L)OSS programs can be done by

i by monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines,

m because all these methods will bring external current (F/L)OSS program knowledge into the organization and thus make people aware of the existence of (F/L)OSS.

o reducing uncertainty about (F/L)OSS quality,

i by using a (F/L)OSS maturity model

m because this will provide an objective measure of the quality of a (F/L)OSS project and thus help assess if the (F/L)OSS program is fit for purpose.

Individual factors

The individual ideology of an IT manager is a motivator that creates a driving force for adoption of (F/L)OSS in the form of a (F/L)OSS champion (Glynn et al., 2005). The opposite also appears to be true. When managers have a “lack of visions on (F/L)OSS business models”, this causes problems with the adoption of (F/L)OSS in a public administration setting (Ozel et al., 2007).

Two theories that explain this are found in Miralles et al. (2005) and Morgan and Finnegan (2007). Miralles et al. (2005) explains why CIOs do not always decide in the overall interest of the organization, but according to a different set of individual objectives (Laffont & Martimort, 2002). In situations where CIOs are “facing complex decisions with incomplete information they tend to rationally run with the pack”(Miralles et al., 2005) and this is called informational cascading. These managers observe the decisions of other managers “without the full knowledge of the reasons why

those decisions were made" and when bounded in the decision making process they tend to imitate other organizations in the "same strategic group" (Miralles et al., 2005).

Reputational herding is the name for concerns that IT managers have about their reputation (Kauffman & Li, 2003). Morgan and Finnegan (2007) show how perceptions of (F/L)OSS of an IT manager can work against adoption. For instance CIO's do not want to be associated with having chosen the "loosing platform" (Miralles et al., 2005). They will thus adopt only a system that the majority of managers have chosen, regardless of evidence that a non-conventional decision is in the best interest of the organization (Miralles et al., 2005). This is a method for the managers to avoid risk (Hamel & Prahalad, 2005). And also happens in non-adopter environments, where there is "a lack of interest in adopting (F/L)OSS" (Miralles et al., 2005).

Theory-based design principle 9

c In the context of complex IT decisions with incomplete information,

o a managers choice based on rationality and in the best interest of the organization is created,

i by changing the perceptions that managers have of (F/L)OSS and providing them with information about (F/L)OSS,

m because this will prevent informational cascading and reduce reputation herding. Knowledge is gained and thus the urge to imitate the decisions of other managers is reduced. Furthermore taking away irrational fear of damage to the managers reputation will cause them to decide more rational.

4.1.2 Architecture policy

An IT architecture is the "organizing logic for applications, data and infrastructure technologies, as captured in a set of policies and technical choices, intended to enable the organization's business strategy" (Feeny & Willcocks, 1998). It is a plan that can be used as a tool for aligning IT and strategy (Feeny & Willcocks, 1998; Eilander, 2008).

Creating an IT architecture involves three steps:

- Define the organization's strategic objectives.
- Define key IT capabilities for enabling those objectives.
- Define the policies and technical choices for developing the IT capabilities.

The strategic objectives of a government organization are: spend as little public money as possible (as seen in previous section) and they have the "obligation to build sustainable and transparent systems" (Ghosh, 2010).

The "key IT capabilities" to implement the strategic objectives (lower cost, sustainable and transparent systems(Ghosh, 2010; Laszlo, 2007; Glynn et al., 2005)) are interoperability, flexibility and transparency. In order to reduce costs, an integrated IT system is necessary (Masrek et al., 2009). Integration means that parts of the total system need to be able to connect easily in order to share information (interoperability) (Wheeler, 2004; Ghosh, 2010). When the business process changes, the IT system must be cheap to adjust in order to reflect these changes in the IT system. That means the total system must consist of parts that can be easily adapted and replaced (flexibility) (Woods & Guliani, 2005; Feeny & Willcocks, 1998; Carbone & Architect-Nortel, 2007). In order to allow the interoperability and flexibility to be sustainable, transparency of the underlying technology to achieve interoperability and flexibility is necessary (Kamphuis, 2009). That way control is guaranteed over the technology and it is thus sustainable in the future.

Therefore the technical choices a government should make are to use (F/L)OSS and open standards. Because the characteristics of (F/L)OSS are such that interoperability, flexibility and transparency are achieved by default. Next to this choice, policies should be created to guarantee that the three principles are guarded. Failure to cement these principles into an architectural policy can lead to destruction of investments in open source, because it makes an organization very vulnerable to opportunistic behavior.

A good example of this is found in (Glynn et al., 2005). This was a case study at Beaumont Hospital. This hospital introduced a lot of open source technology in 2003. Among others were StarOffice (predecessor of OpenOffice.org) and it created its own (F/L)OSS solution to view and distribute x-ray images. The article is very optimistic about the amounts of money that would be saved and the good example the hospital is setting. However in the annual report of 2008 (Hospital, n.d.) it can be read that all the 1100 desktops installs of StarOffice had been replaced by Microsoft Office. And that over 500.000 documents in open document format had be reformatted to the proprietary Microsoft document format. On a national level the director of ICT for health services had made a 40 million Euro purchasing agreement for 35 hospitals[1] for a proprietary system to view and share x-ray images (Medicexchange, n.d.). If there had been an open IT-architecture policy on local and on national level, this relapse would have been much more difficult to rationalize.

Theory-based design principle 10

c In the context of acquiring (F/L)OSS

o protecting (F/L)OSS investments can be done,

i by creating an IT architecture based on (F/L)OSS and open standards,

m because defining the basic principles of (F/L)OSS that are in line with the strategy of the organization in a formal policy at a high level in the organization will make it difficult to argue for proprietary systems that are not in the best strategic interest of the organization.

4.1.3 Procurement process

In this section the literature about procurement of (F/L)OSS is investigated. There is not much literature about this subject. Procedures have yet to be developed (Larsen, Holck, & Pedersen, n.d.). Four recent documents however have made propositions. These specialize in the "procurement" of (F/L)OSS. This whole section draws on the following sources (Ghosh, 2010; Centra, 2006; Daffara, 2008; Eilander, 2008).

Tactical purchasing

As was shown in the research questions section, the most interesting part for (F/L)OSS about procurement happens in the tactical purchasing part of the process (please refer to figure 3.1 for a catch up). The literature found about (F/L)OSS procurement focuses on this part as well.

Because it is impossible to "procure" (F/L)OSS (it is free and *libre* thus can be downloaded (Oosterbaan, 2010)), when the "procurement of (F/L)OSS" is being stated what is meant is the procurement of an IT solution based on (F/L)OSS created by a system integrator, or the procurement of services that deliver knowledge for the implementation of (F/L)OSS (Ghosh, 2010). This knowledge can then be used internally to implement a (F/L)OSS solution.

[1] The first hospital that will receive the new system is Beaumont.

In both cases (integrated (F/L)OSS solution or knowledge services) a conventional procurement process can be followed. For governments specifically this means tendering, which is an extremely expensive, time consuming and complicated process (Oosterbaan, 2010). When tendering for (F/L)OSS especially the formulation of the requirements needs attention and methods to determine the costs are ROI and TCO (Ghosh, 2010; Woods & Guliani, 2005).

Determining specification

One of the biggest disadvantages of the process is the specification phase. Because when procuring large, complicated and integrated systems (e.g. enterprise resource planning systems) it is nearly impossible to deduce and capture all the needs of an organization (Rosemann et al., 2005; Oosterbaan, 2010). The requirements' elicitation process might even require the work of expensive external consultants, making the process even more costly (Oosterbaan, 2010). All these expenses occur even before a comparison between options can be made. Next, governments must neutrally specify the requirement, without asking for a brand or technology (Ghosh, 2010). This calls for special attention when formulating requirements. Requirements consist of three forms: functional-, technical- and business-requirements (Ghosh, 2010).

Functional requirements Functional requirements describe the purpose and functionality that is expected of the software. They should be formulated in a way that addresses the need in the problem domain. It is also the place to state justifications for the need for open source. For instance the agency might wish to comply with the constitution[2] and thus ensure transparency of government processes. Any and all software in use by the government can be considered in this light. But an obvious example is the central voting computer system.

The same goes for functionally specifying open standards. It is essential for a government to be able to let citizens fully interact with their IT systems, without forcing any preference for a particular software or hardware vendor upon the citizen. It is also paramount that the government stays in control of its data. Having the data locked away in a proprietary standard by default turns over this control to the vendor.

Technical requirements Technical requirements specify constraints and/ or needs regarding the technology that is offered. This is where the importance of an IT architecture comes into play. If an open IT-architecture has been agreed upon *before* the tendering process, this makes it easier to state that the new software must be compatible with this open IT-architecture. Failure to implement such an open IT-architecture policy, can harm this reasoning. For instance, if there is any proprietary standard or software stated in the IT-architecture, asking a vendor to be compatible with this will only deepen the vendor lock-in.

Therefore in case there is no *open* IT-architecture, it is better to state the technical consequences of (F/L)OSS. Namely the need to be able to modify and study the software (or have any third party of its choice modify and/or study it) now or in the future, in order to make the software work with other software and/or in case it needs to be adapted to future needs.

Business and service model requirements Business and service model requirements form the specification of the methods of how the government wants to pay and account for the software. This is determined by the architecture and the organization of the agency.

[2] Article 111 of the Dutch constitution says: "The government in the execution of its tasks tries to be overtly in accordance with the law."

Governments are free to discriminate against business models. It is perfectly normal to specify, for instance, that it wants to buy company cars and thereby shut out lease businesses. The same logic follows for (F/L)OSS. Licences for software is a business model that does not have to be respected by governments. Simply put: "There is no obligation on the part of a public body to adapt its requirements to the preferred business models of particular firms" (Ghosh, 2010).

A valid justification for not wanting a proprietary business model is, for instance, the wish of the government to distribute the software internally (or to others) with which it interacts, with no additional costs based on the number of users (per seat licensing model).

Selecting a vendor

One can either become one's own vendor or select a *system integrator*. This is because a (F/L)OSS solution usually involves more than one program. A system integrator will configure the different (F/L)OSS programs according to the requirements. And in the case that the internal IT department will configure and integrate the (F/L)OSS programs, it might be necessary to procure additional knowledge and support from (F/L)OSS specialists (Ghosh, 2010). In these cases a procurement process can be followed. If a single (F/L)OSS program is needed, than "just downloading it" is easiest (Oosterbaan, 2010).

Sometimes when selecting between system integrators, Return On Investment (ROI) and Total Cost of Ownership (TCO) calculations are used. The specific methods to conduct a ROI or TCO calculation vary (Ghosh, 2010; Knubben, 2004; Woods & Guliani, 2005). Furthermore these calculations can't be compared and have to be made for each specific case (Woods & Guliani, 2005). It is worth the effort to make these calculations, but be sure to include the long term costs and exit costs.

Long term costs are difficult to assess. For instance (Ghosh, 2010, page 22), the life cycle of a proprietary operating system might be three years. After these three years a new licence must be bought. In a normal TCO calculation this new licence would not be included since it is a cost incurred after the end of life. But the organization still requires operating system functionality. The (F/L)OSS case is different. Support for the operating system might be bought. This fee will stay the same more or less, but the organization might learn in the mean time. So after five or ten years less support could be required.

Exit costs should be taken into account too. In the example above it was mentioned that a new proprietary licence "must" be bought. Contractually this is not the case. In theory one is free to chose another operating system after the three years. However lock-in might have occurred. Meaning that costs will be incurred if choice is made to switch to an alternative. Ghosh (2010) calls these *exit costs* and they belong to the legacy technology. So in determining the costs of a proprietary software product, these *exit costs* (to move away from the lock-in) should be included in the price of a proprietary system. The characteristics of (F/L)OSS make lock-in impossible, so these costs will be near[3] zero for a (F/L)OSS solution.

Contracting

This phase consists of two parts when dealing with (F/L)OSS. First, if a system integrator or a support contractor is chosen, then agreements with these vendors will arrange the "specific commercial and legal terms and conditions" of the deal (Weele, 2005). Second, when downloading and implementing (F/L)OSS yourself or purchasing integration services that create a (F/L)OSS solution,

[3] Because if (F/L)OSS is used without open standards, converting the documents to another standard might come at a cost.

registering the software within the organization is a best practice (BSA, n.d.). This is of course not to prevent piracy, but (F/L)OSS licences are still licences and the organisation must be aware of the terms (Ghosh, 2010). Next to this it might be convenient for management to know how much and what kind of software is used. Also, unauthorized downloading and installing of software has great security risks for the IT infrastructure. So keeping control is a good business practice.

Theory-based design principle 11

c In the context of (F/L)OSS services procurement,

o (F/L)OSS knowledge or a custom build integrated (F/L)OSS solution can be obtained,

i by using a procurement process that uses specific terminology in the requirements to specify the (F/L)OSS characteristics,

m because governments should specify neutral requirements and this terminology will make sure the (F/L)OSS characteristics are translated to valid neutral requirement statements that will request (require) (F/L)OSS without asking for it specifically.

4.2 Practice-based design principles

With a multiple-case study the practice-based design principles are developed in this section by a within- and cross-case analysis. First, the results of the six within-case analyses are given. These consist of the most important lessons learned from each case and provide a structured overview of the challenges and the solutions found in each of the cases. Then the practice-based design principles will be extracted by a cross-case analysis. The cross-case analysis will use the within-case analysis as input in order to extract the practice-based design principles.

4.2.1 Within case analysis

Each within-case contains a case description and two analyses. The short case description introduces the case, but focuses mostly on highlighting the noteworthy situations of the case. The within-case analysis resulted in two tables. One table gives an analysis of the challenges, issues and solutions that were found in the case, the so-called “solution table”. And the second table shows the process that was followed in the case, the “process table”. To save space, only the solution tables are displayed here and the process tables can be found in appendix A. In order to quickly give an overview of the cases and for further reference, table 3.2 provides some important case characteristics.

The Dutch patent office

The Dutch patent office (DPO) is an agency of the Department of Economic Affairs in the Netherlands. It has around 130 employees and they maintain about 150 thousand patents currently registered in the Netherlands. Of those employees, 10 are working at the IT department of the DPO (NOiV, 2010).

One of the reasons for the DPO to have an active interest in (F/L)OSS is that the state secretary of Economic Affairs was looking for organizations that were willing to execute a pilot to demonstrate a working open desktop. He needed this for his action plan. This led to a business case which concluded that an open desktop at DPO was possible and a project was formed.

The issue most at play in this case was the disentanglement of the existing IT products that were in use. In this project there was a lot of room (time, money) to experiment with (F/L)OSS given

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been doen about it?*	*Is there a long term solution?*
1	New techniques have to be learnt.	I have been working like this for year. It's already working. I'm one year from retiring.	Giving Linux training.	
2	Worn-in habits.	The change made some ineffective work processes visible	The processes were adapted	
3	(F/L)OSS has a bad image	Hobbyist, "spielerei", free so can't be good. unknown = unloved.	Showing that (F/L)OSS works	
4	People are afraid of change	Three types of people 1) Advocates 2) As long as I can do my job 3) do we have too?	Explaining how the system works. Showing efficiency. Letting the users choose. Experiment with (F/L)OSS.	
5	If you want to replace one thing, you need to replace the whole stack.	Lockin causes high costs to get rid off. Use (web)standards.	Actively search for alternatives. Hybrid environment. Look at the ICT systems from an architectural viewpoint.	Take the de-integrate costs into account when making a TCO.
6	Watch out for per seat licenses. You don't have to prove it's the vendors' fault when applying for support.	per seat does not scale. Goal is a different business model in the market. Current SLA's are best effort. (F/L)OSS supplier is much more focussed on providing solution. (F/L)OSS project involves many small vendors. Any software company can help.	Different contracts "subscriptions". Pilot gave room to choose "real" open source.	
7	Difficult to explain how (F/L)OSS contributes to efficient work flow. Costs of the proprietary licences play a role.	Can (F/L)OSS help me do my job quicker? Transparent government. Reason in action plan are valid but the "business" reason not so very clear. Proprietary licences are expensive.	Can be defended by resorting to agreed upon policy. Costs are driven down by choosing technology based on different business models.	
8	Cloak and tender agreements can hurt these kinds of initiatives. They reduce flexibility and autonomy of IT-department to conduct the project.	These contracts are complex and impossible to bypass. They also can form lockin.	Nothing can be done. You need to follow the rules. The challenge is how to create conditions in these contracts that prevent vendor lock-in.	

Table 4.1: Summary of the strains and coping interventions in the Dutch Patent office case.

by top management. So a close look was taken at all the software that was used at the agency. The conclusion was that nearly all the software was so integrated and entangled, that they had to replace all the main systems.

Together with two (F/L)OSS expert firms, the DPO agency created a project plan with what, how and when to replace. The goal was to use (F/L)OSS wherever possible. For this they developed a specific methodology (see A). The companies also helped the DPO to make the replacements. Final decisions were left to the end users. First a long list would be created to scan for relevant (F/L)OSS options. Than a short list (usually three alternatives) would be selected according to very specific wishes and then the final choice was made by the end users, based on small one-day pilots.

Department of Defense

The case took place at IVENT. This is an organization within the department of defense that delivers the department's IT services. One of the products they maintain are the around 60000 desktop deployments throughout the Dutch military apparatus. The case revolves around a new "Internet At the Workplace" service, which is implemented by using (F/L)OSS. The Research and Innovation Center (RIC) of IVENT (the R&D department), engineered the new service and the IT maintenance department had to deploy it.

However the existing skills of the IT maintenance department were incompatible with the skills required for the new (F/L)OSS implementation of the service. A maintenance team from a different product group (UNIX services), was able to support the new solution, but this type of service was formally not their responsibility. This caused friction. A senior innovation manager of the RIC summarized the issue: "The people that were meant to maintain it could not, and the people that could, were not allowed to". After discussions, the final solution was to let the maintenance team with the most compatible skill set implement the service.

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been doen about it?*	*Is there a long term solution?*
1	The old situation was expensive. Open source could save money and demonstrated a better performance.	Licences of the closed source product were user based. This made the old situation expensive and had performance issues. There was no money to expand the licenses, although demand was high.	modularity saves money Implementing open source reduced the licenses costs.	
2	The people that were supposed to maintain the innovation could not. While the people that could were not allowed to.	There was a transfer of an innovation from R&D to maintenance department. The latter did not have the capability to support the innovation.	Transfer to another department. Making 2 products. The old one became IODW plus and then new fun one.	
3	From different parts in the organization there was resistance to the new system	New browser People liked the new service Confusion about who was supposed to do what (my job to give people a browser)	Explaining how the new browser worked End user evaluations (got an 8 out of 10 being the best) End user representative in the production project	
4	There were issues with "not invented here" syndrome.	There was fear of (F/L)OSS. And foot dragging (took a year still no action).	There was top management support. There was policy to use (F/L)OSS. Both combined triggered with the foot dragging caused a response to work only with the willing. The RIC showed initiative in developing the (F/L)OSS solution.	
5	There were a lot of changes at the same time. Expanding the old environment, upscaling the new one, a reorganisation and new hardware.	It was an innovation project. Knowledge needed to be transferred. There was a reorganization	Letting people that were willing and possessed the knowledge take out the project. Showing that the new product worked, was save and maintainable	

Table 4.2: Summary of the strains and coping interventions in the DoD case.

Case Z

This case is classified as confidential. Therefore only general statements are made here to prevent the case being traced back. The case involved introducing new features to a legacy environment that was operating in the primary business process. The introduction of these new features was behind on schedule and the project was started for the purpose of expediting the implementation. The conditions of the case were such that the scope (in time, money and technology compatibility) was limiting the option to use (F/L)OSS. In the end proprietary software was used as a solution.

However they did investigate what the impact would be on the future possibilities for using (F/L)OSS, despite not opting for (F/L)OSS now. It was found that the lock-in situation was not getting worse and that future (F/L)OSS solutions were possible. One of the methods to ensure that was the use of open standards. That way, the new connections that are needed for the features are not in a proprietary standard. Thus the system at either side of the connection can be easily replaced by alternatives.

The Dutch tax office

The Dutch tax office is a large organization in the Netherlands (30000 employees). They take care of over 6 million tax returns of Dutch citizens each year. Within this organization there is a department that takes care of the applications that are deployed to the department's 30000 standard desktops.

This department noticed a rise in demand for Adobe writer, a proprietary product used for desktop publishing purposes. The department did not understand the rise in demand since the core business of the Tax office is not desktop publishing. After an investigation, they found out that there actually was a need for a conversion utility to transform documents from their proprietary office product into pdfs.

In this case a clear entrepreneurial effort was displayed by the team implementing the (F/L)OSS solution. They started their own investigation on their own initiative, they convinced the internal customer to use their (F/L)OSS solution and then they helped change the standard internal pro-

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been doen about it?*	*Is there a long term solution?*
1	There was no ambition (top or bottom) to do anything with open source in this project. People can always frustrate a process if they do not want it.	In other projects at this organization ambition from the bottom up was present. There open source succeeded. People need to be motivated to do something different.	The government tries to motivate people to consider open source. Internal R&D organizations invent ways to use open source. This reduces the research time and knowledge needed.	
2	The combination of different roles that people have provides an obstacle. How can you make sure that system administrators and developers will self evidently chose (F/L)OSS?	People act in different roles. Not all roles are not designed with the (F/L)OSS paradigm in mind. Dutch tender laws are not build for (F/L)OSS. People want to perform their role in the best way that they can.	Roles were used as defined by the process method and normal procedures.	(F/L)OSS system integrator service providers should be challenged by procurement personal to take part in the tender.
3	Strict deadline. Project needs to be operational within a year.	No time to experiment or get experience with other software.	Choosing for proprietary software made a perfect fit with the legacy environment and thus saved engineering time.	
4	Strict budget.	Costs were easy to calculate. The gains were more difficult to calculate. No budget for a large overhaul that would be needed if open source was chosen.	They had to use the resources at their disposal. But they looked at what was needed for replacing the legacy environment.	A larger project with wider scope can get the necessary funds. (F/L)OSS reduces costs in sharing a solution, because the (F/L)OSS solution can be freely copied.
5	Little autonomy.	Not a pilot. Not a green field situation. No options or room for experiments.	Implementing proven off the shelf technology made sure the scope was met. Greenfield or pilot situations are ideal for (F/L)OSS adoption.	
6	The project had to adjust a legacy proprietary environment ***. Certain projects are so large that it's impossible to be independent of vendors.	Existing maintenance service organization that wants to maintain the current situation. Costs will rise when choosing a different new product due too lack of knowledge. Choices of the past for related technology cause restricted room for (open source) alternatives.	Research was conducted to find out if this current choice for proprietary technology would limit the future use of open source technology. Special agreements can be made to check vendors and hold them accountable. This could also be used in the (F/L)OSS case.	A research project with greater scope and more resources can be used to change the proprietary legacy environment to a new open one.
7	This was a highly visible project. To big too fail.	Large and wide impact on daily operations of the organization.	Making sure all the tender procedures are followed very closely. Document the procurement process.	
8	Some tender policies cause more harm then good. If policy is not set, a lot of time is lost in discussions about details.	Policy that is set, makes discussions easy. The reasoning that every company deserves a fair chance can be a strain.	There already was policy that the project should have this form.	Have a more consolidated approach. Look at architecture and build policies based on operational needs, architectural possibilities and strategic long term vision.

Table 4.3: Summary of the strains and coping interventions in the Z case.

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been doen about it?*	*Is there a long term solution?*
1	Downloading was forbidden, because of security reasons, but downloading had to be done to be completely (F/L)OSS licence compliant and thus legally save.	(F/L)OSS is seen as a product. Incompatible procedures to acquire a (F/L)OSS program.	During the research phase, ordering a cd from a vendor.	Introducing new policy that formalizes downloading.
2	Registering software with the price value of zero was impossible.	It's best practice to register software that is used. Paradigm: a product cannot be free.	First time by ordering a cd and paying for that service. This left the necessary paper trail. The second time by a new procedure.	In a new procedure the legal department organizes a contract and register number for the software on their own. Therefore the software can be registered.
3	Doing things differently causes extra work. That work has to be accounted for.	This was a new situation. And colleagues needed to act differently.	Explaining the situation and convincing them to help out.	
4	The word "open source" was controversial. Sometimes naming that word could stop a project on a management level.	People that are convinced that (F/L)OSS is a bad thing. Sometimes (F/L)OSS is misunderstood. (F/L)OSS is unknown.	Being careful how to formulate a project proposal.	
5	(F/L)OSS has a bad image.	There is no marketing for (F/L)OSS. But there are many FUD stories around. Unknown to larger audience.	Communicating what (F/L)OSS is. Showing that it works. Learning and building on experience.	
6	Building software yourself is not considered an option.	Although DTO has the capabilities, media and other stories, convince decision makers that this option is "always" more expensive, but it isn't.	Showing that this can be done.	

Table 4.4: Summary of the strains and coping interventions in the Dutch tax Office case.

curement procedures in order to make the deployment of (future) FLOS-Software officially possible within the DTO.

Grootegast municipality

Grootegast is a village in the north of the Netherlands in the province Groningen. It has around 12000 residents and the IT department of the village manages around 80 active desktops per day. Two factors were noteworthy in the case. The display of foresight and the strategic positioning of the IT department.

The foresight is demonstrated by the fact that Grootegast adopted the policy to use (F/L)OSS several years ago in 2004. Around that time research was done to see if it was technically possible to start using (F/L)OSS (A7 report (Gijtenbeek & Malipaard, 2004)). The conclusion was that a solution was possible, but that this solutions was rather expensive in maintenance, so the municipality chose not to start with it then.

However, because they were still interesed in using (F/L)OSS, in 2008 the IT department looked again at the possibility. This time, in light of the formal end of life of the proprietary office product that they were using, it was established that the (F/L)OSS solution was easy to maintain by making use of a third party proprietary tool. This cleared the way to start implementing the (F/L)OSS solution, together with the maintenance tool.

The strategic positioning of the IT department originated from the realization in the municipality that the IT Department played a pivotal role if Grootegast wanted to seriously offer "digital services" to the residents. This resulted in the entire organization being restructured. Instead of separate columns with the IT department as one those columns, the organization is now oriented towards delivering services to residents. At the basis of these services resides the IT department. Today the IT department is involved in many of the municipality's decisions since in the end it is they who have to deliver the services to the residents as well as the internal customers.

This position within the organization creates the necessary space for the IT department to take a long term architectural view of the organization's whole IT infrastructure. For instance, the experience gained of how to deploy OpenOffice.org is now used in a neighboring municipality in order to deploy OpenOffice.org there as well.

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been doen about it?*	*Is there a long term solution?*
1	Employees need to support the new system. There could be fear for the new system.	If the employees don't support the system, it will not be used. Or people will try to frustrate the introduction.	This can be solved by showing employees what is happening. Give more functionality in the new solution then they had in the previous. Give training. Pilots. And hold an evaluation after the introduction.	
2	Project planning might fail. Trust may disappear.	Any change in the organization introduces stress.	communicating with the employees. Teaching. Reserve budget for change management. Communicate a clear vision and stick to it.	
3	Discussions and misunderstanding might undermine the project.	People might deliberately try to sabotage the project.	By being in a central position in the organization, discussions about open source are minimized by creating understanding. A central position also creates autonomy.	
4	Failure of the widows terminal server 2008 made OpenOffice.org look bad, because it appeared that OpenOffice.org crashes instead of Windows. The project should not appear to come from the IT department.	One must guard the image that open office gets in the organization.	This was solved by communicating and explaining what is happening. And fixing the bug in Softgrid.	
5	The IT department in Grootegast are very ambitious, which causes a lot of work.	They execute a lot of research. Try to do everything themselves. And they watch what happens in the open source communities of relevant projects.	Making sure the ICT is more centralized, so they can achieve an economy of scale. That way they have more time for all the ambitious projects	
6	Large software vendors have a lot of power over the organization.	These vendors have a monopoly position and hold data hostage. Some vendors therefore gain a lot of power in the organization.	Talking to them and showing them that more municipalities are switching to OpenOffice.org. Explaining that the law says they should use open standards.	
7	Lack of top management support will not give you the necessary autonomy and resources.	The head of the IT department is an influential figure in the organization and also holds other positions. Planning ahead made sure there was already policy to introduce open source more easily	By making sure the decision is cast in policy, no discussion can appear. The policy was already decided. So there are strategic reasons instead of operational reasons to execute the project.	
8	The reasons for open source must be of a strategic nature.	This will make sure resources stay committed if the project is not getting along.	Communicating the reasons and have them formalized in policy.	
9	Lack of foresight can cause compatibility problems in the future.	One needs to be aware of the organizational changes and the technological changes in the environment.	Grootegast knew they would switch to OpenOffice.org some day so they kept track of the development of the project.	

Table 4.5: Summary of the strains and coping interventions in the Grootegast case.

Voerendaal municipality

In the far south of the Netherlands lies the village Voerendaal. It's about the same size as Grootegast (12000 residents). Two people make up the IT department that takes care of around 80 desktops. The IT department has the goal to outsource as much of their IT as possible. No servers are maintained at the site for instance. The two employees of the IT department provide the first and second level of support to the end users.

Voerendaal changed many parts of its IT infrastructure. Together with the choice for new office software, it changed the house style, buying new PCs and after that also new server hosting service. This big change was carefully planned. Before it implemented OpenOffice.org it held a thorough investigation to determine the software options available. Users appeared to like the OpenOffice.org user experience more and therefore this was implemented.

Its main proprietary back-end application caused a lock-in situation, because it could not connect with OpenOffice.org. This was resolved by creating a piece of software that took care of the connection and releasing this software as (F/L)OSS. After a year of using OpenOffice.org, the vendor of the proprietary back-end had also created a coupling for OpenOffice.org.

4.2.2 Cross-case analysis

The practice-based design principles were extracted by making a cross-case analysis. The basis for the cross-case analysis is the within-case analysis of the cases. The within-case analysis delivered two tables per case as was shown in the previous section. In this cross-case analysis first the process tables from each within-case were compared and analyzed. This resulted in two practice-based design principles and are presented in the "process comparison" section. Second the solution tables were compared. Six themes emerged from that analysis and each of these themes resulted in a practice-based design. These are presented in the "solution comparison" section.

Process comparison

In this section a comparison is made between the process tables from the within-case analysis to see what the practical "procurement" process for (F/L)OSS looks like. It therefore answers the first research question: "How do organizations deal with (F/L)OSS programs when procuring software for their operational processes?". The main conclusion is that the steps that were taken in each case look more like a project than the expected procurement process. Also most of the interviewees were talking about "their project" and not a procurement process. They also expressed their worries when "procurement people" were involved in making decisions related to (F/L)OSS.

What can be generally seen across the cases is that "procuring" (F/L)OSS is basically done in two steps. First the decision is made to use (F/L)OSS, based on some proof of principle or strategic arguments. Then the actual choice of software is governed by a project. This can be seen in table 4.7 that displays the steps taken in all the cases next to each other.

Looking at the decision stage, it becomes clear this is actually a make/buy decision. Although the cases here did not "make" any (F/L)OSS software, configuring it to make it work and integrate it into the existing infrastructure was all done in-house. And this was an extensive exercise.

Looking more closely at the project stage, three project strategies can be identified: flying under the radar, operating on an island, operating in a greenfield (for an overview see table 4.8. Each strategy has a different scope, goal and is applied in different situations. Besides these strategies, all the cases used extensive project planning to guide the project to success.

#	Challenge	Issue	Coped	Resolved
	What is the challenge?	*What are the arguments and reasons?*	*What has been done about it?*	*Is there a long term solution?*
1	Natural upgrade point.	Usability of OpenOffice.org was found to be better than Office 2007. ODF compliance at the end of 2008. A new house style was also needed.	All these factors combined caused Voerendaal to investigate (F/L)OSS possibilities and finally introduce OpenOffice.org.	
2	Several lockin situations exist.	The CBS organization demands a yearly dump of the financial situation in a proprietary format. OpenOffice.org had to be linked to the proprietary back-end. And third party server based computing suppliers appeared only to offer lower costs when choosing for proprietary server OS.	Much work went into making a connection with the proprietary back-end. Than a year later the proprietary vendor started to (partly) support OpenOffice.org. The other lock-ins have not been resolved. For the CBS dump, Voerendaal uses one separate proprietary office license just for that task. And they had to choose proprietary servers because of pressure from the proprietary back-end supplier and the external server hosting service.	
3	Proprietary office licenses were expensive.	By introducing OpenOffice.org Voerendaal could drive down costs. However open source server hosting was found to be more expensive than proprietary server hosting.	They switched to OpenOffice.org to meet their office software requirements. And because of the cost savings that could be made by choosing proprietary server OS, they opted for that.	
4	To reduce ambivalence about the course of action top management support was needed.	The head of IT was ambitious about open source.	The IT department got approval for their plans from the city council. And the city also joined the list "Open overheidsorganisaties" which expressed their long term desire to be more open. One of the reasons for these actions was the national policy "actie plan NOiV".	
5	Some people had doubts and were unfamiliar with the new software.	As with any new software, also OpenOffice.org requires training.	By making use of the "teach the teacher" principle the IT department managed to educate the employees within 2 weeks. They also provided an usb stick to all the employees so they could use the software at home too. For instance when they wanted to work on a document after work.	
6	"Big bang" change, so highly visible	They changed a lot at the same time (servers, house style, OpenOffice.org)	The project was planned carefully. The team made sure that they had a grip on the technology and predicted issues that might arise. This gave them a chance to be prepared for challenges. Furthermore the team had much autonomy to operate at their will. The neighboring municipality (Vaals) could give tips, because they already use (F/L)OSS. The project included pilots to find out any issues end-users might have before the final release.	

Table 4.6: Summary of the strains and coping interventions in the Voerendaal case.

DTO	Grootegast	DPO	Case Z	Secint	Voerendaal
Motive	First research	Motive	Motive	Motive	Reason
Research	Motive	Business case specifications	Business case	First improvement	Research
	Research	Business case selection	Impact analysis	Research	Research
	Presentation to the management team	Business case execution			Implementation of servers
Selection	**Choice**	**Choice**	**Project proposal**	**Decision**	**Pilot and approval**
Contracting	Deliberate with suppliers	Project plan what to replace	PID	Implementation	Implementation and second pilot
Acquire	Pilot	Thinking phase started about how to replace	Implementation		Implementation
Registering	Implementation	Implementation phase started execute			
	Training				
	Evaluation				

Table 4.7: Comparison of the processes of the cases.

In the next sections, first the decision stage is examined further and then the three strategies and project planning from the project stage are explained.

Configure versus procure In the investigated cases the IT departments created their own integrated solution (especially case DoD and DPO). That is why the "procurement process" in the cases looks like a project. The customer is designing or, more accurately said, "putting together" (Olivier Sessink) their own solution by using generic off the shelf "as is" (F/L)OSS components. This is not completely making it yourself, but much more the configuring[4] of their solution.

Within the cases only a kind of "mini procurement" took place. The IT-staff actively went looking for different (F/L)OSS programs with the same function, then downloaded, tested and compared them on a small scale to find out which one met the requirements best and could thus be selected and used. In most cases pilot schemes were also used to find out if the end-users like the system. This is an advantage of (F/L)OSS because it meant that no procurement was needed: "for a lot of stuff we did not have to bother the procurement department [..] that saved time" (Olivier Sessink, DoD).

In all of the cases a preference for off the shelf software was stated. During the selection of software, programs are evaluated on the functionality they currently have ("as is"). There are reasons for this OTS preference and there are reasons to look beyond the current functionality, to the *potential* of a (F/L)OSS program.

(F/L)OSS programs can be considered as off the shelf. The development of (F/L)OSS programs is a continuous process, but development is frozen at particular points in time (a release) (Dinh-Trong & Bieman, 2004). This release version can be considered the off the shelf equivalent of (F/L)OSS. A reason not to change the source code to fit your organization's needs is that the organization "will need to retain that knowledge of the source code in house. And we already almost don't have that knowledge. The less change, the cheaper the long term maintenance costs" (Olivier Sessink DoD).

However, when changing the source code, there is the opportunity to take part in the (F/L)OSS development process of the program by giving the changes back to the community (policy in DPO, Grootegast, Voerendaal case, minor[5] in the DoD case). If the changes are accepted by the project, the organization does not necessarily have to take care of this change. Because it will be maintained by the community of the project.

This scenario was discussed in the DTO case. The interviewees thought that DTO was (unfortunately) not making use of this characteristic of (F/L)OSS, because managers consider "making" to be always more expensive than buying, while this does not necessarily have to be the case. The next example was discussed in the DTO interview.

If a (F/L)OSS solution was found, but it is for instance not in Dutch, than it would be turned down completely, even though the application fills all the other requirements. For (F/L)OSS it would be relatively easy to create a translation when the organization already has compatible programming capabilities[6], which they have at the DTO. So the (F/L)OSS program could be translated and the translation committed to the project. Maybe someone will use it, maybe not.

Furthermore the way support was seen is different in the cases. The organizations are not dependent on the support of a vendor. Instead, because technically everyone owns the source code, any company can become expert in the software and start delivering support and other solutions that might help. This increases competition and thus lower prices for support. And since the only thing

[4] This configuring is needed in proprietary products as well depending on what kind of implementation is done (Rosemann et al., 2005).

[5] Some configuration scripts or forum posts

[6] Actually in some cases no programming skills are required depending on the i18n technology used. Sometimes only a translation of a list containing all the menu options needs to be made and saved

the support vendors can compete on is how quickly and well they can solve problems, the relationship with the support vendor changes dramatically. It reverses the burden of proof. One is not receiving support with the product, but receiving support *as* a product.

Because of (F/L)OSS some cases (Grootegast, DoD, DPO, DTO) took a second look at their support contracts and found out that when a vendor is contacted for support, the vendor has only the obligation to make an effort to find a solution and not the obligation to give you a solution. Sometimes the vendor will search for excuses not to deliver support (DTO case, where Adobe blamed PDFcreator while there clearly was a bug in their proprietary product). The situation is demonstrated by the following quote : "The only option that those contracts give you, is that you can hold them accountable and that you might get a financial restitution. [...] the end situation does not change. In either case[7] you still do not have the solution that you wanted in the first hour" (Arjan Duinkerk, DoD). Of course holding your vendor accountable also involves some risk, especially if the vendor has a monopoly. It might be a bad strategy to have a quarrel with that vendor.

So (F/L)OSS seriously blurs the difference between making and buying. This is not a choice to make lightly. "When no change of procurement and development is planned, the management might have not understood the scope of change required for the adoption of (F/L)OSS" (Daffara, 2008). What becomes clear from the cases is that there are three options. One is downloading (F/L)OSS and implementing it yourself. The second is procuring by tendering for a system integrator that will configure a (F/L)OSS solution on the basis of requirements. Thirdly, downloading (F/L)OSS yourself and implementing, adapting or configuring it by procuring the support of a (F/L)OSS knowledge specialist.

Practice-based design principle 1

c In the context of a (strategic) choice for (F/L)OSS use

o implementing (F/L)OSS can be done,

i by downloading and installing and configuring (F/L)OSS yourself,

m because when there is enough knowledge of (F/L)OSS, this is possible and (F/L)OSS allows anyone to do this and it is possible to by-pass a procurement department.

i by procuring a (F/L)OSS solution from a system integrator,

m because there might not be enough internal knowledge about (F/L)OSS or the organization wishes to outsource IT, then having a system integrator configure the solution is an option.

i by procuring the support of a (F/L)OSS knowledge specialist to assist with the downloading and installing and configuring of the (F/L)OSS solution,

m because there might not be enough internal knowledge about (F/L)OSS but the organization wants to internalize this knowledge in order to become self (reliant) proficient with (F/L)OSS.

Flying under the radar strategy Flying under the radar means you start small, gather support and then replace a critical system. The risk limitation is based in evolution (baby steps) towards an implementation (DoD and DPO case).

The strategy works as follows: the project starts out as small (beta, proof of concept, pilot). Then when the project proves its functionality and adds value for users, it wins over the hearts and minds of the organization (see 4.2.2) This catches any opponents off guard and thus can create friction. To handle this, there is a need to implement change management as seen in the section 4.2.2.

Flying under the radar limits risks, because the project grows as long as the project is still supported from within the organization. First few resources are necessary to get the proof of concept in play and only when the solution demonstrates the usefulness and the organization supports it, a choice

[7] He compared (F/L)OSS project support with their current SLAs

can be made to roll it out officially.

In the DoD case it delayed discussions about the IT skills incompatibility that would have to stopped the project before it began if the project proposal was discussed with all stakeholders. In the DPO case end users where informed about the plans, but could not intervene. Only the final choice (between (F/L)OSS programs) was given to the end-users (see section 4.2.2).

So the flying under the radar tactic is an extremely effective strategy to get rid of mission-critical legacy systems, but can create friction in certain parts of the organization. A second disadvantage is that resources (especially (F/L)OSS knowledge) already need to be available (for the first proof of principle) which could require top management support.

Operating on an island strategy An island operation means one finds a possible isolated area to introduce (F/L)OSS and then do it. The area needs to be non-critical to the business process (e.g. only used on a small scale). The island tactic works when one does not need or have a lot of resources from top management and your entrepreneurial spirit finds an opportunity to use (F/L)OSS.

If the project succeeds a precedent will have been created for the use of (F/L)OSS, if it fails it will have a low impact and thus limit the risk to the business process and prevent damage to the (F/L)OSS image (DTO case).

In the DTO case the reason to use (F/L)OSS was to lower the costs. The use of (F/L)OSS in this case caused the necessary changes in the procurement processes (see also section 4.2.2) to ease the use of (F/L)OSS in future cases. This can be a reason why one might consider this tactic.

Operating in a greenfield strategy Operating in a greenfield is the third tactic and involves introducing completely new functionality. Therefore nobody will notice if the project fails, because the functionality is not there in the first place and thus won't be missed. Next to that, because the functionality does not exist yet, it offers room to experiment without having a large impact on the organization (DPO and Voerendaal).

If the new functionality needs to integrate with a legacy environment that uses proprietary standards, it is difficult to introduce (F/L)OSS. In such a case replacing the legacy environment to get in line with architectural guidelines *before* starting to introduce new functionality is better (see section 4.2.2 for architecture). However this might be expensive, especially if the legacy is in the primary critical business process. This was exactly the situation in case Z.

When the new functionality does not have to integrate with a legacy environment, the introduction of (F/L)OSS is easy. This strategy was not present specifically in the investigated cases, but the interviewees did mention it. One example was mentioned in the Voerendaal case where they gave the new city council members a laptop running a (F/L)OSS operating system. They also had a couple of laptops for giving presentations that where using a (F/L)OSS operating system. Another example was given in the DPO case:

> "For instance just right this morning, I get a call from another agency, they had seen our wiki that we use as a knowledge base. 'We like it. Is that expensive? Can we do that too?' [...] So they called me [with those questions], well the license is just regular GPL, you can download it from the Internet and install it. You just have to look how much time you want to spend tweaking it. It's that simple. [...] it took us half a day to install, no not even half a day less, and then you can use it." (Sander Mittertreiner, DPO)

When using this tactic, lobbying for organizational support or implementing some change management is still needed to make the adoption by the end users easy.

Type of action:	DPO	DoD	Case Z	DTO	Grootegast	Voerendaal
Flying under the radar strategy	x	x	-	-	x	-
Operating on an island strategy	-	-	-	x	-	-
Operating in a green-field strategy	(mentioned only)	-	-	-	-	x (laptops)
Project planning	x	x	x	x	x	x
Actively search for (F/L)OSS ("mini procurement")	x	x	-	x	x	x

Table 4.8: Strategies applied in the cases

Project planning A method carried out in all the cases was project planning. Project planning appears to be absolutely of the essence. It makes sure the endeavor is thought through and its impact is assessed. This will set the scope of the project. Planning for contingencies will make sure risks can be limited.

In the cases people planned to manage for organizational support, and they investigated what exactly to replace and why. This was in order to make guestimates about technical issues, issues with interconnections and in order to determine the scope and necessary resources (time and money) for the project. In all the cases that were seen here, the people planned for an innovation. For instance whereas replacing MS office with a new version is fairly straight forward and can be automated, migrating to OpenOffice.org is an entirely different case[8]. Even the relatively stand-alone dropin replacement in Voerendaal was carefully planned. They conducted pilots, wrote new software for the interconnect with the back-end and planned for educating end users in the form of teach the teacher. The planning reduces the risk of surprises.

Practice-based design principle 2

c In the context of replacing a business critical system yourself,

o implementation of (F/L)OSS is achieved,

i by downloading and installing in parallel a (F/L)OSS solution with the same functionality as the operational business critical system and gradually evolving this (F/L)OSS solution until it is mature and proven to replace the current system,

m because the evolving approach requires beta releasing which can build organizational support for the solution. It also minimizes risk and can demonstrate that it's functionally sound to replace the system.

c In the context of a non business critical process,

o creating a precedent for the use of (F/L)OSS can be done,

i by downloading and implementing the (F/L)OSS solution yourself,

m because this method does not require a lot of resources from top management, but entrepreneurial spirit of the implementer will find an opportunity to use (F/L)OSS.

c In the context of introducing completely new ICT functionality yourself,

o implementation of (F/L)OSS is achieved,

i by downloading, installing, experimenting with the (F/L)OSS solution and finally lobbying for organizational support or implementing some change management,

m because the functionality does not exist yet. So experimenting can be done without risk until the solution works. Then to gather organizational support to make adoption of the solution easy, lobbying and change management is required.

c In the context of downloading and implementing (F/L)OSS yourself,

o securing project success is done,

i by planning the project extensively

m because there are a lot of factors to consider. What those factors are depend on the specific situation. Time should be taken to investigate what must be changed and when. That way guestimates about technical issues, issues with interconnections and scope and necessary resources (time and money) can be determined for the project.

[8] Once in place, the maintenance task of (F/L)OSS programs decreases because of the modularity and development model of (F/L)OSS ("Organizations that use (F/L)OSS have 66 PCs per IT administrator, compared to 53 PCs among non-users" (Ghosh, 2005)). For instance rolling out a new OpenOffice.org version can be completely automated. If it fails it can be rolled back without any problems. Because the organization is in control of the application *and* of the interconnections.

Case	Entre-preneur-ship	Top man-agement support	Lock-in	Archi-tecture	Organizational support	Change manage-ment	Not classi-fied
DPO	No	Yes	Yes (5,6)	Yes (8)	Yes (1,3,4)	Yes (2)	7
DOD	Yes (4)	Yes (1)	Limited	Limited	Yes (3,2)	Yes (5)	-
Case Z	Yes (1,5)	Yes (3,4)	Yes (6)	Yes (8)	Unknown	Yes (2)	7
DTO	Yes	Yes (4)	Limited	Limited	Yes (5,3)	Yes (1,2)	6
Grootegast	Yes (5,8)	Yes (7)	Yes (6)	Yes (9)	Yes (1,3,4)	Yes (2)	-
Voerendaal	Yes (1)	Yes (4,3)	Yes (2)	No	Yes (5)	No	6

Table 4.9: Inducing themes from cases.

Solution comparison

In this section the solution tables of the within-case analysis are compared. It therefore answers the second research question: "What problems are found in these organizations when acquiring (F/L)OSS programs and have they found solutions for this?". Table D.1 shows the link between parts from the solution tables of the within-case analysis and the 6 emergent themes. The table is the result of an iterative process that reduced all the challenge-solutions sets from the within-case analysis, by looking for common themes. From these themes, the practice-based design principles have been distilled. The following sections show this, by elaborating upon each theme extensively to demonstrate the evidence for the corresponding practice-based design principle.

The table works as follows, the numbers in the table correspond with challenge-solution sets taken from the solution tables in the within-case analysis. For instance, challenge-solution set number 4 from the DoD case, deals with an issue relevant to entrepreneurship. In the next sections, each of the themes is further explained and per theme is shown in which cases the theme was present. See appendix D for an elaborate commentary on the table.

Entrepreneurship Entrepreneurship is seen in 5 of the 6 the cases. In every case there was a group of ambitious people that identified an opportunity to solve an IT challenge with (F/L)OSS. The (F/L)OSS road was chosen, because the group was truly convinced that the organization would be better off with a (F/L)OSS solution.

Choosing for (F/L)OSS can be considered an entrepreneurial step, because there is no reward for using (F/L)OSS and no penalty for choosing proprietary. The easy, risk-averse action is to go with the proprietary technology (case Z). If the people in the cases had chosen to acquire proprietary products, nobody would have stopped them, because there is no checking to see if it was possible to use (F/L)OSS in these situations (DTO).

Next to that, traveling the (F/L)OSS road means taking the road less traveled, because it requires changing procedures outside your normal operating environment in some cases (DPO, DTO, DOD). It thus requires lobbying and convincing colleague's in the organization to help implement the solution (see 4.2.2).

The people in the cases were motivated mostly by the national policy. The action plan was mentioned in all the cases as the trigger to start looking for (F/L)OSS, sometimes despite what top management might have thought about it at that time (DTO).

(F/L)OSS allows for easy experimentation, because it requires few resources to get it working, especially when compatible IT knowledge about (F/L)OSS is available. The people in the cases *already knew* about existing (F/L)OSS solutions and because (F/L)OSS can be freely downloaded it becomes easy to experiment and provide a proof of principle. This also takes away any doubts the (F/L)OSS

champions might have about the feasibility of a (F/L)OSS. For instance in the DTO they investigated a (F/L)OSS solution to the problem, thereby overcoming their own fear[9] of (F/L)OSS. But before this can happen, it must be known that (F/L)OSS exists. Educating them about alternatives reduces the "what the eye doesn't see, the heart doesn't grieve over" (DPO,DTO) challenge.

Practice-based design principle 3

c In the context of downloading and implementing (F/L)OSS yourself,

o stimulating the entrepreneurial attitude towards (F/L)OSS of employees, can be done,

i by pointing out the national policy and the (F/L)OSS philosophy,

m because this serves as a trigger to start searching for opportunities to use (F/L)OSS.

i by giving employees room for experimentation,

m because this will make it easier to determine if a (F/L)OSS solution meets the requirements.

i by educating them in different kinds of (F/L)OSS alternatives,

m because this will increase the chance opportunities are identified to use (F/L)OSS.

Top management support Top management support is found to be a key issue in all the cases. For instance the DPO case: "Commitment from the management team is everything. If you don't have that, than you can forget about it . . . it won't work." (Sander Mittertreiner, DPO). This is because top management can make sure there are enough resources committed and can motivate the organization.

In some cases (Voerendaal, Grootegast, DoD, DPO) top management support was already available because the management knows and is convinced about (F/L)OSS. The opposite is also true. When a top manager is convinced that (F/L)OSS is bad, project proposals will be ignored (DTO). In either case, the management does require some sort of proof.

If top management support does not already exist it has to be created by convincing the top management with arguments. In the cases three arguments were used for this purpose:

1. Compliance with national policy (NOiV).
2. Presenting a successful business case or proof of concept.
3. Cost calculations, showing (F/L)OSS will save money.

The first reason was used in all cases. The second was used in the DPO case, where an external company did a business case, the DoD case created a proof of concept, case Z did impact studies, Voerendaal and Grootegast did a pilot and checked the feasibility of OpenOffice.org. Only the DTO case was a bit different since they did not need formal approval other than the customer's agreement. But they too examined for themselves if the solution was feasible.

Costs also played a role in all the cases. All the cases saved money by implementing a (F/L)OSS solution. These savings were mostly on licence costs. In the DPO, DoD and DTO cases per seat licenses were in use. For larger organizations this is rather expensive: "because that amounts to a rather large sum of money very quickly. Consider the *00 thousand Rijks employees times a *00 euro..." (Tjeerd van der Laan, DTO). In the DoD case because of the tight defense budget for instance, the possibility for basic Internet access from all the 60 thousand standard DoD desktops

[9] Illustrated by the quote: "Are we up to the task?"(Andres Baas, DTO)

was severely limited by the total cost of the per seat licences. Grootegast and Voerendaal were at a point were they needed to decide to invest in renewing licences for MS Office. Choosing open source meant either delivering more of the same service (DoD, DTO) or the same level of service for less money (Voerendaal, Grootegast). They also argued that this was fair because it was public money that they were spending (Grootegast, DPO), and the money that can be saved could be used to educate the administrators (DPO).

Practice-based design principle 4

c In the context of acquiring of (F/L)OSS in a government setting,

o obtaining top management support,

i by convincing top management with reasons (compliance with national policy, proof, costs calculations),

m because compliance with national policy and a successful business case shows confidence that the solution can succeed. So a risk avoiding manager is more inclined to agree. And lower costs are usually something managers aspire to by default, especially in government settings since it is public money that is being spent.

lock-in Nearly all the cases had to deal with vendor lock-in caused by technical compatibility (DPO, Case Z, Grootegast and Voerendaal). Only DTO had fewer of these issues, since the (F/L)OSS solution was a near drop-in replacement for the proprietary software. In the DoD case they could introduce a solution next to an existing one and had to deal with existing IT-skills. In this section the situation in the cases about technical and network lock-in is made clear and then their solutions are explained. Dealing with existing IT-skills can be done by change management (see section 4.2.2).

Technical lock-in was found in Voerendaal and Grootegast, because their existing legacy office software was highly integrated with their back-end systems. Without any additional adjustments, replacing MS Office with OpenOffice.org, meant that they would not be able to get the data from their back-end systems into OpenOffice.org. In the DPO case this integration with office and the IE browser was also found for their CMS, CRM and Groupware systems. In case Z, they had the most trouble with lock-in because the planned expansion integrates deeply into the existing legacy system. Because the (F/L)OSS alternative differs significantly from the existing legacy technology, much work would be necessary to make the fit. Therefore choosing a (F/L)OSS solution was deemed impossible, because of the time and financial scope restrictions of the project.

Network extranalities caused two other lock-in situations in Voerendaal and Grootegast: the first one due to the CBS and the other by hosting service vendors. The CBS[10] distributes a spreadsheet in a proprietary format that is too complex for a user to convert to an open standard. This spreadsheet must be filled by the municipality with its financial data. The solution in Voerendaal is to install proprietary office software once a year specifically for this task. In Grootegast they still have some proprietary office versions permanently installed. The second lock-in was experienced by Voerendaal when looking for a hosting service for their servers. The hosting services for (F/L)OSS systems appeared to be more expensive and in less supply than proprietary systems[11]. And Voerendaal's back-end system required proprietary operating systems.

The solution for the technical lock-in caused by the back-end was to force the vendor to start using open standards (Voerendaal and Grootegast) so a connection between OpenOffice.org and the back-end could be made. Voerendaal created a new piece of (F/L)OSS software on their own to make the connection before the solution was available from the vendor.

[10] CBS is a Dutch agency that gathers business intelligence on the government and the Dutch economy for public purposes

[11] This is an example of "prior technology drag" (Kauffman & Li, 2003) see section 4.1.1

In case Z, the technical lock-in was actively investigated to determine if choosing to expand the current legacy systems was going to make the lock-in situation worse and could thus make a possible choice of (F/L)OSS in the future more difficult and expensive.

The technical lock-in in the DPO case was almost completely eradicated, because the previous vendors refused to use open standards. That's why the DPO simply replaced all the proprietary software with (F/L)OSS and open (web) standards as far as they could. This shows that, sometimes because of lock-in, almost the entire software stack needs replacing even when the goal is to only replace one part[12]. That way one does incur sunken costs from the existing products.

Next to this replacement they also actively search for other possible (F/L)OSS solutions and have taken the point of view that a hybrid situation (use proprietary and FLOS-software) is more realistic than completely running on (F/L)OSS. However only some parts of the total solution are still proprietary in the DPO case. And some of those have a licence model which is based on a support subscription. In such a model, money is explicitly paid for support which reverses the burden of proof (as was seen in the previous section). Because in that case one only needs to ask for support, instead of having to first prove that a product is malfunctioning before support is received (DTO, DPO).

The lock-in caused by network externality was not resolved. Voerendaal choose to use the cheaper proprietary operating systems hosting service for their back-ends and the CBS still requires the use of the proprietary document formats.

Practice-based design principle 5

C In the context of vendor lock-in caused by legacy technology,

o reducing lock-in is done,

i by convincing the existing vendors to use open standards

m because open standards make it possible to interchange parts of a total IT solution.

i by replacing as much software that causes lock-in with (F/L)OSS,

m because (F/L)OSS uses open standards and thus makes it possible to interchange parts. And by using (F/L)OSS one does not rely on a specific vendor but instead can get support from many different companies.

i by preventing that current investments will make future investments of (F/L)OSS more difficult,

m because lock-in will at least not be increased by the new investments. This will also prevent these current investments to become sunken costs in the event that (F/L)OSS is chosen in the future.

Architecture One way to make the (F/L)OSS procurement last is by having the fundamental principles translated to an IT architecture (DoD). In this section it is shown that this prevents (F/L)OSS niches (case Z) and helps protect investments made in (F/L)OSS from cloak contracts (DPO).

The cases in this research can all be regarded as niches. When generalizing the cases, it can be seen that groups of people are trying to change the way (F/L)OSS is handled in their organization, sometimes from the bottom up (DTO), sometimes with great help from top management (DPO). But they form a relatively small group of projects where (F/L)OSS is used (it covers only 12 % of end-user oriented (F/L)OSS cases[13] in the Netherlands, see also appendix E)in the public sector. This random creation of (F/L)OSS niches could not be considered a structural approach to "procuring" (F/L)OSS (case Z). A suggested solution to form a more structural approach is to have an architectural policy

[12] This kind of lock-in can be *prevented* by taking an IT architecture approach as is shown in section 4.2.2 about architecture. Here the focus is on the direct ways to deal with *existing* lock-in that were used in the cases.

[13] The actual number might be higher because it is not (yet) policy to track (F/L)OSS use in infrastructure software.

(DoD). This allows for strategic control over ICT use and should be guarded by a representative in top management (DoD, case Z).

That way not only will small groups of ambitious people consider (F/L)OSS, but all departments in an organization will *have to* consider using (F/L)OSS. In some from this is already happening, but only on a voluntary basis. The NOiV action plan is the most cited reason for getting started with (F/L)OSS next to cost reduction. One can consider the NOiV action plan as a kind of architectural policy. The problem with this plan is that there are no consequences for not following this plan it. Furthermore it is to general to use for a specific organization.

What is needed is a specific set of rules (and exceptions) for the organization in question and a way to make sure these rules are followed. This can be done by making use of "architectural models" (DoD). Using these will make sure there is a process in place that checks if new IT projects confirm the architecture as set forth by top management (Case Z and DoD). To demonstrate what happens when this architectural model is not enforced: "at our organization this [the architecture] is not secured (enforced), so if a project leader thinks he is gonna do some politically incorrect things, then he is definitely not gonna do something with architecture" (DoD).

The creation of these rules is difficult. Too strict rules and people will be unable to follow them. If they are too lax, they won't make a difference. There is also need for valid exemptions from the rules. For instance the software to control x-ray machines in an emergency room might not comply with a (F/L)OSS based IT architecture (DoD). But doing without an X-ray machine is not an option. So valid exceptions from the architectural rules are necessary. These exceptions can be allowed for only a limited time. So an architectural policy should deal with these three things: rules, exceptions and enforcement (DoD).

The second benefit of an IT architecture is that they will protect investments in (F/L)OSS from public tenders. According to European and Dutch law, the government must make use of tender procedures in some procurement cases. These tender processes are found to be very cumbersome and expensive. The unwritten policy in practice is to avoid them as much as one can. And if one really must tender, then to make sure so called "mantel" (cloak) contracts are agreed upon.

These contracts cloak specific details. For instance instead of tendering for toilet paper, the cloaking deal would mention "household consumables" or even more vaguely "facility services". The point is to make the term vague enough so it can cover a wide range of products. This way the tendering process is avoided, which saves time and money.

These contracts can form a problem in "small initiatives like the one at DPO" (Tjeerd van der Laan). In the DPO case there were no cloak contracts for software, but for people. It was initially not a problem, later a public procurement was made for open source expertise. This expertise was not covered by the contracts that were in place. So the company delivering the people with the expertise easily got the tender: "For a certain project we officially won the tender" (Emiel Brok, DPO).

But the personnel department of the DPO found out about that together with the current vendors of personnel. So the contracts were changed to incorporate job descriptions instead of expertise. This meant that the open source expertise vendor had to stop the work and compete with the other tender partners for each separate assignment. This was very inconvenient.

When an architectural policy based on (F/L)OSS principles is in place, it can guard from cloak contracts. This is because it does not matter what kind of procedure is used to implement (F/L)OSS. In a procurement operated by a tender process, the external system integrator partner will have to comply with the architectural model of the organization. And when a project is started inside the organization to implement software, the employees too will have to adhere to this architectural plan.

So formulating a sound IT architecture and maintaining it can help the procurement of (F/L)OSS. Top management of an organization must clearly make a decision what to do regarding (F/L)OSS. This is a strategic decision. One that only top management has the required legitimacy to make and enforce. After the IT architecture is implemented it can be used as a tool to strategically steer on issues like costs and vendor dependence (DoD).

However an IT architecture is not without problems. As was mentioned, the creation of the rules and their exceptions is tedious. Furthermore the architecture needs to be maintained and enforced at a high level in the organization. And finally until the procedures are in place and work, this method will only cost money (DoD).

Practice-based design principle 6

c In the context of acquiring (F/L)OSS,

o implementing an architectural IT policy will structure the strategic application of ICT,

i by creating an architectural ICT policy guarded by top management and supporting processes in the organization,

m because this will put a price on non compliance to the architecture and thus will make sure employees have the necessary motivation to create room in projects or procurement's and thus choose technology that is strategically beneficial to the organization.

Organizational support In the investigated cases a change introduced into an organization (i.e. using (F/L)OSS technology) needs support from the employees before it can be adopted. It was found that this organizational support was created by taking away the fear, uncertainty and doubt around (F/L)OSS. This is done in four ways. First the image of (F/L)OSS is improved by explaining. Second showing that the (F/L)OSS technology works, by offering pilots and have the end-users have a role in the final decision. Thirdly by communicating the status of the change and what is being done. Fourth by giving them training in the technology after it has been implemented.

(F/L)OSS appears to have a bad image. This is because people are not familiar with the phenomenon. First of all people may simply be unaware that (F/L)OSS exists. In the second place (F/L)OSS might be misunderstood.

In cases (DPO, DoD) it was mentioned that "what the eye doesn't see, the heart doesn't grieve over". This shows that even on the IT departments, not everybody is familiar about (F/L)OSS technology. This makes it hard to find (F/L)OSS programs that might solve the IT challenges.

When there is awareness of (F/L)OSS, there can be misunderstanding about the concept. This is because there are a lot of rumors about the (F/L)OSS concept that are taken to be true (see for instance (Oosterbaan, 2010). To solve this in the DoD case for instance this uncertainty was removed by executing extensive security risk analysis. This "informing and explaining" showed that (F/L)OSS was not different in security aspect as proprietary software (and even safer in some cases).

Pilots can help tremendously to build organizational support. A clear example of this was found in the DPO case. In this case a long list was created on the basis of a general characteristic (CRM functionality). Then a short list was created only on the basis of must have additional functionality. The final choice between the systems on the short list was left to the end-user. The three final candidates were simply installed to let the end-users "play with them". Because nearly all the end-users got a vote on the final system this created a high level of organizational support for the final solution.

Organizational support is also organized by keeping the rest of the rest of the organization informed about the progress. In Grootegast for instance the PR department held interviews with the pilot groups and communicated the results on the intranet so all the employees could see read about

the pilot users' experiences. In the DPO case, the people executing the change kept repeating the reason why this change was being done. This has the consequence that employees are informed about the progress and thus reduces fear of the unknown new technology.

Also training helps foster organizational support. Because training will make end-users able to use the technology, thereby taking away any uncertainty and doubt about the technology. In all the cases training sessions were given to help end-users get familiar with the new systems. The term end-user is broad. In the DPO case, the end-users were system administrators who needed to learn how to administer the (F/L)OSS, but also employees that needed to work with the software program. In Grootegast and Voerendaal this training was guided by external training bureau's. This prevented that the whole project was experienced as "an IT party". This advice was also mentioned in the DPO case.

System administrators can also be in need of new knowledge. This knowledge came in the case of DPO from outside of the organization and in case of the DoD, from the internal R&D department.

> "One of the most important aspects I think is having the right people [with knowledge of (F/L)OSS]" (Sander Mittertreiner, DPO)

> "Speaking very generally, what we have seen is that if you want to do a lot with open source, than almost always simple basic knowledge of Linux is very convenient. [...] so if you strategically want to steer towards (F/L)OSS use, you need to start educating employees." (Olivier Sessink, DoD)

In Voerendaal and Grootegast, system administrator knowledge was less an issue. The administrators that needed to maintain the final solution, were also the once implementing it. So that meant they learned how to administer the system along the way.

Quite the opposite was true for the DoD case. Their R&D department, created a proof of concept that was intended to be transferred to the production environment where system administrators would maintain the service. However the system administrators that were responsible for this type of service, did not have the required knowledge to support (F/L)OSS. This resulted in a view of the new technology as competition, instead of a new tool they could use to provide a better service for the end-users.

This situation led to discussions to resolve the issue. Eventually tasks were redivided. a different group of administrators from the production environment took over the maintenance of the new technology. These administrators had experience with related technology (UNIX) so it was easier for them to acquire the new knowledge (and some already possessed the knowledge next to being enthusiastic about the new technology).

Practice-based design principle 7

c In the context of downloading and implementing (F/L)OSS yourself,

o creating organizational support can be done,

i by reducing misunderstanding by explaining what (F/L)OSS is and informing them about the possibilities

m because showing what exists will make employees understand and familiar with (F/L)OSS

i by making use of the (F/L)OSS characteristic that easily a fully working pilot setup can be installed one can let the end-users of the organization in on the final IT decision,

m because letting end-users in on the decision of the final solution will generate the feeling that they took part in the project which will create internal support for the final solution.

i by communicating continuously with your end-users,

m because this can explain the situation and goals and will thus reduce the fear for the solution.

i by training end-users in the new technology,

m because this will show that the solution works and thus reduce the uncertainty and doubt about the new technology.

Change management In the cases three categories can be found. One were employees work routines using the system needed to change (DPO), second changes in ICT procurement procedures (DTO). And a changes in existing roles (case Z).

In the DPO case a lot of systems were changed. End-users were trying to apply their known ways of working on the new systems. Because this did not always work, it became clear that some people had been using inefficient work habits. Training and showing them more efficient ways sometimes by making use of new features of the new system resolved this. Next to this, they also had to train and motivate their system administrators to support the new technology. For instance: "At some point a system administrator said hey I just recompiled my kernel at home to get WIFI working! Then I think, well that guy made a giant leap in knowledge level and excitement" (Tjeerd van der laan, DPO)

In the DTO case they had to adjust the existing procurement procedures in order to make the "procurement" of the (F/L)OSS program officially possible. There were two problems, registering the new software and downloading it. Normally a proprietary product is procured from a vendor. This generates a legal contract which can be registered in a management system. Then a physical box arrives with the software on CD or DVD which can then be deployed.

However the (F/L)OSS solution did not have a vendor in the regular sense. That meant that a contract between vendor and buyer could not be made and thus also not registered. Furthermore the legal department had studied the (F/L)OSS license of the solution and concluded that it had to be downloaded. This in order for DTO to be compliant with the terms in the (F/L)OSS license and thus be save in a legal sense. But downloading software was prohibited at the DTO.

Both problems were solved by implementing new policies. This took some effort on the part of the team to convince others: "You are only bothering people from the [other] services. When you are doing this, it is something new and then they have to do something different and that means more work and change for them" (Andres Baas). But this was not opposition *per se*. "You are dealing with something new. That means some things will take longer or require research. But that's not resistance" (Jan van Kommeren). It was additional work that just needed to be done. And this required some convincing to make it possible.

From Case Z we learn that existing roles can be an inhibitor of change. For instance "result driven project managers" will try to reduce the scope of a project and not extend it. A system administrator is focused on one thing only. Making sure the *status quo* is maintained. It's his job to make sure the system (and by extension the service) as it is is up and running. "Those people will almost per definition put their foot down. And I can't even blame them. It's their job". Procurement officers must comply with Dutch and European procurement laws. "And those have not been fit out for (F/L)OSS, on the contrary they are fitted out for big companies that have profit accountability. [...] So you will have to do the procurement differently". All these people need room in order to do their tasks differently.

Practice-based design principle 8

c In the context of acquiring (F/L)OSS

	tbdp 1	tbdp 2	tbdp 3	tbdp 4	tbdp 5	tbdp 6	tbdp 7	tbdp 8	tbdp 9	tbdp 10	tbdp 11	Final design principle (fdp)
pbsp 1	-	-	-	-	-	-	-	-	-	-	x	fdp 1 Acquiring (F/L)OSS
pbsp 2	-	-	-	-	-	-	-	-	-	-	x	fdp 1 Acquiring (F/L)OSS
pbsp 3	-	-	-	-	x	-	x	x	-	-	-	fdp 2 Entrepreneurship
pbsp 4	x	-	-	-	-	x	-	-	x	-	-	fdp 3 Top management support
pbsp 5	-	x	-	-	-	-	-	-	-	-	-	fdp 4 Lock-in
pbsp 6	-	-	-	-	-	-	-	-	-	x	-	fdp 5 Architecture
pbsp 7	-	-	x	x	-	-	-	x	-	-	-	fdp 6 Organization support
pbsp 8	-	-	-	-	-	-	-	-	-	-	-	fdp 7 Change management

Table 4.10: This table shows which practice-base design principle (pbdp) was combined with what theory-based design principle (tbdp to form the final design principle (fdp)

o creating an environment ready for (F/L)OSS adoption is done,

i by adjusting work habits

m because the new system might require different ways of performing tasks or offers more efficient ways of performing these tasks, adjustment of work habits might be necessary.

i by adjusting procedures to accommodate (F/L)OSS procurement,

m because current procedures might not be compatible with certain characteristics of (F/L)OSS. Therefore adjusting the procedures by taking into account these characteristics will help (F/L)OSS procurement.

i by adjusting existing roles,

m because the tasks performed in the roles can be the opposite of what tasks are required to introduce (F/L)OSS. Changing the tasks of a role with (F/L)OSS procurement in mind will make sure people are free within their roles to procure (F/L)OSS.

4.3 Design principle synthesis

This section presents the final design principles. It does this by synthesizing 11 theory-based design principles and 8 practice-based design principles to formulate an integrated set of 7 grounded final design principles. For this synthesis the method of Burg et al. (2008) is used (see section 3.4 for an explanation of the method). In the previous section the results from the cross analysis gave answers to the first and second research question. This section answers the third research question: how can the "procuring" be improved? This is done by formulating a final set of design principles that can be used to (re)design the tactical purchasing policies to include (F/L)OSS.

In this section only a summary of each design principle is shown which explains the basics of the design principle. In table 4.10 an overview is given of what theory-based design principles (tbdp) are combined with what practice-based design principles (pbdp). The rest of the chapter will demonstrate the functioning of each of the design principles. A formal description in CIMO logic of the full design principles can be found in appendix[14] C. Also note that the management summary presents a summary of the final design principles in a convenient format.

4.3.1 FDP 1: Acquiring (F/L)OSS

In the context of a strategic choice for (F/L)OSS use, implementing (F/L)OSS can be done,

- By downloading, installing and configuring (F/L)OSS yourself, by following one of the strategies depending on the context,

  when in a business critical context: gradually evolve the (F/L)OSS solution in parallel with the existing system, until the system meets the requirements. Then start migrating users when the (F/L)OSS solution is mature and has proven itself.

[14] In the appendix the design principles are presented in a format that shows the origin of the interventions and mechanisms by referencing to cases and literature. This is not shown in this section!

when in a non-business critical context: just download, instal and configure the (F/L)OSS program. No special interventions.

when introducing new ICT functionality: experiment with different (F/L)OSS alternatives then lobby for organizational support[15]

in all cases: planning the project extensively.

- By procuring a (F/L)OSS *solution*[16] from a system integrator, by using a procurement process in which uses specific terminology in the requirements to specify the (F/L)OSS characteristics.
- By procuring the support of a (F/L)OSS knowledge specialist by using a procurement process in which uses specific terminology in the requirements to specify the (F/L)OSS characteristics. This knowledge specialist will assist with the downloading, installing and configuring of the (F/L)OSS program or solution.

This design principle shows the process that can be followed when a (strategic) choice for (F/L)OSS use has been made. (F/L)OSS programs can be downloaded from the projects website (and thus bypass a procurement department) and then integrated (installed and configured) by the organization itself. This depends on the knowledge that is available from the IT-staff. If there is enough knowledge one of the three interventions can be chosen depending on the context. The fourth (planning intervention) should always be executed.

The first intervention works because evolving the system in parallel requires beta releasing to users (pilots). This can build organizational support. Evolving also minimizes risk by demonstrating the solutions capability. The second intervention works because no resources are required besides the ones already under direct control. Implementing (F/L)OSS will generate precedent for (F/L)OSS use. The third intervention works because the functionality does not exist yet, so experimenting can be done without risk until the solution works. Then to gather organizational support to make adoption of the solution easy, lobbying and change management is required. The fourth intervention works because the factors to consider depend on the specific situation thus time should be taken to investigate what must be changed and when. That way guestimates about technical issues, issues with interconnections, scope of the project and necessary resources (time and money) can be determined for the project.

If knowledge of (F/L)OSS is lacking, or there is a policy to outsource the integration of software programs, then services from a company can be procured by means of a procurement process. This company will then download and integrate the (F/L)OSS program(s) for the organization. The organization can also opt for the possibility to hire (F/L)OSS experts. These experts will then help the IT-staff to integrate the (F/L)OSS programs. This way (F/L)OSS knowledge is procured, which is then transferred to the IT-staff of the organization.

Both procurement interventions work because governments should specify neutral requirements and using terminology based on the (F/L)OSS characteristics will request (F/L)OSS without asking for it specifically.

In both procurement interventions work because governments should specify neutral requirements and using terminology based on the (F/L)OSS characteristics will request (F/L)OSS without asking for it specifically.

4.3.2 FDP 2: Entrepreneurship

In the context of adoption of (F/L)OSS by downloading and implementing (F/L)OSS yourself, stimulating the individual entrepreneurial attitude towards (F/L)OSS of employees and obtaining personal effort, employee commitment and triggering adoption can be done,

- by stating a precise definition of a strategic intent for instance by pointing out the national NOiV policy and (F/L)OSS philosophy, but being flexible in means of how to reach this goal.
- by organizing for a high perceived capability for solutions, by giving employees room to experiment and educate employees in different kinds of (F/L)OSS alternatives by monitoring (F/L)OSS repositories, hiring (F/L)OSS experts, sending key employees to conferences and obtain subscriptions to trade magazines.

[15] Remember this is also created by leaving the final decision for a system to the end user.

[16] A solution exists of more than one software program. Thus a system integration service is procured, not a product.

This design principle has two interventions. Setting strategic intent (goal) will motivate (trigger) employees to get committed to a cause and start searching for opportunities to use (F/L)OSS. This is because current resources will not suffice to reach the goal, so the organization is forced to be innovative. By creating a high perceived capability for solutions the organization makes sure the employees also know what to do in order to reach this goal. This is very important in cases were there might not already be compatible (F/L)OSS knowledge at the IT-staff. This has to be created first, before entrepreneurial efforts can emerge.

4.3.3 FDP 3: Top management support

In the context of radical, high-risk initiatives such as acquisition of complex FLOS-Software in a government setting with incomplete information about the decision, obtaining necessary resources such as time, space, equipment, people and communicating, supporting and sharing, vision and goals is done

- by creating top management support. This is done by changing the perceptions that managers have of (F/L)OSS and by providing them with information about (F/L)OSS. This can be done by pointing out that a (F/L)OSS choice would comply with national policy, show a successful proof of principle and/or business case. And by making cost calculations. And in these calculations account for the switching costs when vendor lock-in is deepened by opting for a proprietary alternative.

Top management support is essential. This is because only top management can provide the necessary resources. This final design principle shows interventions how to convince top management that (F/L)OSS should be given a chance. This can only be done if top management is in the position to make a rational choice. Changing the perceptions that managers have of (F/L)OSS and providing them with information about (F/L)OSS will prevent informational cascading and reduce reputational herding. Knowledge is gained and thus the urge to imitate the decisions of other managers is reduced. Furthermore taking away irrational fear of damage to the managers reputation will cause them to decide more rational. A successful business case or proof of principle will prove that the solution is technically viable. Also offsetting the true cost of proprietary software against (F/L)OSS provides arguments to create top management support. Because if by opting for a proprietary alternative vendor lock-in is created, this will limit the choice when new investments in software are made. The costs incurred moving away in the future from this proprietary software should be added to the current costs of the proprietary product.

4.3.4 FDP 4: Lock-in

In the context of technical compatibility issues (vendor lock-in) with legacy technology, reducing the lock-in is done,

- by convincing the existing vendors to use open standards.
- by replacing the legacy systems by installing (F/L)OSS (alternatives) as much as possible.
- by encapsulating the parts that cannot be replaced, by using middleware and virtualisation techniques.
- by preventing that current investments will make future investments of (F/L)OSS more difficult.

There are two ways to deal with vendor lock-in. First one is prevent it from happening. Second if lock-in exists, reduce it. The first approach requires an analysis of current investments to find out if they will cause vendor lock-in. This intervention could be combined with the architecture design principle. In that case the proprietary product is dissected to see if it can couple (via open standards) with the architecture. The second approach consists of three interventions. Vendor lock-in is mostly caused by the use of "*de facto*" or proprietary standards. Using open standards enables the organization to replace a program or product without much difficulty. So convincing current vendors to use open standards already reduces vendor lock-in. The second intervention is rather simple. Replace the legacy system by a (F/L)OSS program. Or at least a product that uses

open standards. Third intervention is used when the vendor lock-in is so strong that the proprietary product and standard has to be maintained. Then two techniques (middleware and virtualisation) can push back the reach[17] of this proprietary product.

4.3.5 FDP 5: Architecture

In the context of acquiring (F/L)OSS, implementing an architectural IT policy will structure the strategic application of ICT and protecting (F/L)OSS investments can be done,

- by creating an IT architecture policy based on (F/L)OSS and open standards guarded by top management and supporting processes in the organization

This design principle works best in a context were top management has set strategic goals and these goals are translated into an IT architecture. Creating this IT architecture, will cause two things. It will structure the use of ICT in the organization and will protect investments made in (F/L)OSS. The use of ICT will be structured because all decisions now have to fit into this grand IT design. And top management is in control of this design. This means top management is able to steer strategically, for instance give the house wheels so it can move, should the environment demand such a change. Investments in (F/L)OSS will be protected if the IT architecture is based on using open standards and (F/L)OSS programs. Lower levels cannot replace an existing (F/L)OSS system with a proprietary one, because that would conflict with the architecture.

4.3.6 FDP 6: Organizational support

In the context of downloading and implementing (F/L)OSS yourself, creating organizational support can be done,

- by reducing misunderstanding by explaining what (F/L)OSS is and informing employees about the possibilities by monitoring (F/L)OSS repositories, hiring (F/L)OSS experts, send key employees to conferences and subscribe to trade magazines.
- by increasing trialability by letting end-users play with the solution. This can be done because it's easy to setup a working pilot with (F/L)OSS programs and letting end-users make the final decision.
- by communicating continuously with the end-users by showing that the solution creates benefits for them and sharing experiences and results. This increases observability and takes away fear.
- by training end-users in the new technology.

This design principle can be applied to create organization support for (F/L)OSS. It relates the technology adoption factors to the fear, uncertainty and doubt that can exist in the practical field about (F/L)OSS. There are four interventions. The first one is meant to get the correct idea of (F/L)OSS across in the organization. The second intervention is used to let the end-users partake in the decision process. This reduces fear since they can see the program working and the partaking creates a feeling that their comments are taken seriously. The third intervention is especially useful when there is a pilot in progress. Sharing the experiences of the pilot end-users will lower any uncertainty about the program. The fourth intervention is necessary so that the end-users can use the program and thus experience that the program helps in their performance of tasks.

4.3.7 FDP 7: Change management

In the context of acquiring (F/L)OSS, creating an environment ready for (F/L)OSS adoption is done,

[17] e.g. an essential product that keeps track of pencil use, only runs on a proprietary OS. Then installing this product on a virtual server running the proprietary OS and distributing it via open (web)standards (terminal server), prevents the use of the proprietary OS as the primary OS for all the desktops in the organization. The desktop OS can then be replaced by an (F/L)OSS OS

- by adjusting work habits.
- by adjusting procedures to accommodate (F/L)OSS procurement.
- by adjusting existing roles.

This final design principle is only based on a practice based design principle. It was found that a change to a new ICT system can require a change in tasks of employees. Three interventions were found. The first intervention works because the new system might have more efficient methods to perform a task. This thus requires a change in work habits. The second intervention can be used if the current procurement process is incompatible with certain (F/L)OSS characteristics. One must realize that a change in procedure might be necessary to procure certain (F/L)OSS services. The third intervention can be used to create room for employees to act upon (F/L)OSS. For instance a system administrator tries to maintain the current IT situation. Changing this role by allowing him to also experiment, will create an opportunity to try out (F/L)OSS programs.

5 Discussion

This master thesis had two goals, generating new knowledge about the 'procurement' of (F/L)OSS by government organizations. And developing design principles that would help government organizations (re)design their 'procurement' policies to allow for (F/L)OSS. In order to achieve these goals the following research questions were posed: how do government organizations deal with (F/L)OSS, what are the challenges and how can the 'procuring' be improved. These research questions were answered by formulating a set of design principles grounded in theory and practice. These principles form guidelines that can be used to create solutions for how to deal with (F/L)OSS (Romme & Endenburg, 2006). Developing these design principles also generated new knowledge about the 'procurement' of (F/L)OSS by government organizations.

The first contribution of this research is the insight (shown by a design principle) that (F/L)OSS cannot be procured. But it is possible to procure integration services and (F/L)OSS knowledge. The second contribution are two strategies to implement (F/L)OSS in an organization, that emerged during this research. These are "building niches" and the "coordinated effort" approach. The third contribution are the remaining design principles that can be used to create an environment which makes (F/L)OSS adoption easier. The following sections explain the three contributions further.

5.1 Downloading (F/L)OSS and procurement of services

The "acquiring (F/L)OSS" design principle shows that (F/L)OSS cannot be procured. In none of the cases, procurement of (F/L)OSS programs took place. This is because of the special characteristics of (F/L)OSS programs. Notably that (F/L)OSS programs are *libre* and have no vendor. Therefore they cannot be procured. (F/L)OSS integration services, (F/L)OSS support services and (F/L)OSS knowledge (in the form of personnel) however, can be procured. Because these are conventional services provided by vendors. This realization gives the procurement process a logical place. Namely the procurement process can be used to either procure integration, support or knowledge services. It cannot be used to procure (F/L)OSS programs.

5.2 Two emergent strategies to implement (F/L)OSS

Two strategies to implement (F/L)OSS in an organization emerged in this research. These are 1) building niches and 2) by a coordinated effort.

5.2.1 Building niches

The "entrepreneurship" design principle shows that ambitious people can drive (F/L)OSS implementation. What was seen in the cases is that (small) groups of ambitious people with compatible

(F/L)OSS knowledge are downloading the (F/L)OSS program(s) themselves and integrating these programs in their organizations ICT systems. These initiatives form niches of (F/L)OSS. There are two driving forces behind these initiatives. First one is the NOiV action plan (NOiV, 2007) (shown in the entrepreneurship design principle) and in second place, the adoption factors are appropriately dealt with (shown in the analysis of the cases). The first driving force causes motivation for people to act and the second makes sure there is an environment *capable* of fostering (F/L)OSS.

Although this works[1], it leaves the final implemented (F/L)OSS solution vulnerable to opportunistic behavior. It appears in practice that (F/L)OSS niches can get into trouble. In the thesis the Beaumont hospital was already mentioned. But also Heerenveen (Heur, 2010), Amsterdam (Rooij, 2010) and München (Schiefil, 2010) are all examples of niches from practice where the (F/L)OSS implementation is under attack, because of opportunistic[2] reasoning and lack of political or strategic policy. This research found a solution for this. Namely a coordinated effort to undertake (F/L)OSS implementation.

5.2.2 Coordinated effort

The architecture design principle shows a way to protect investments in (F/L)OSS by a coordinated effort to (F/L)OSS implementation. In this coordinated approach, strategic (or even political) objectives are formulated at the highest level in the organization, on the basis of business goals. These strategic objectives are then translated into an IT architecture which is guarded by top management. In this IT architecture a choice is made to use (F/L)OSS, because the (F/L)OSS characteristics align with the strategic goals set forth by top management. This way, opportunistic behavior becomes more difficult. Compliance with IT architecture cannot be ignored and there are strategic reasons that will overrule (any) opportunistic operational reasons not to use (F/L)OSS (see for instance (Gelles & Waters, 2010)).

Specifically for a government this can be done. The strategic goal of a democratic government is to create and use systems that are transparent, efficient and sustainable (Ghosh, 2010; Cassell, 2008; Holst, 2010; J. Lee, 2006). As was demonstrated, (F/L)OSS unlike proprietary products can implement all these strategic goals. There are also legal arguments that support a preference[3] for (F/L)OSS, because governments can discriminate against business- and/or development models (Lessig, 2002; Ghosh, 2010; Wijnen-Meijer, 2007). And since (F/L)OSS uses a different development process and requires different business models from vendors, this can be done.

5.3 Planning and adoption

The third contribution are the remaining design principles that can be used to create an environment which makes (F/L)OSS adoption easier. In applying either strategy (building niche or coordinated effort), project planning and an environment that can adopt (F/L)OSS appeared to be vital. The rest of the synthesized design principles in this master thesis can be used as guidelines to create this environment in an organization. From those factors influencing adoption, the most important adoption factor was found to be compatible knowledge. Knowledge about the existence of (F/L)OSS, (F/L)OSS programs and how to use them are vital. It does not matter whether the (F/L)OSS

[1] It is unsure if these niches will work as an oil spill. This heavily depends on creating the right environment in which (F/L)OSS can foster. This is done by paying attention to the adoption factors.

[2] For instance cheaper deal with a proprietary vendor.

[3] This of course only applies when procuring integration services. And specifying that the software used in the integration, should be OSI certified. And any code written to make the total solution work, should be released under an OSI certified licence.

integration is done by the organization itself or if this integration is procured as a service. Knowledge is necessary to maintain the final solution. If this knowledge does not exist, building it within an organization might take a long time. Hiring external (F/L)OSS trained personnel can speed up the diffusion of (F/L)OSS knowledge (Nagy et al., 2010).

5.4 Conclusion

This master thesis report has shown a systematic approach to combine the knowledge from existing literature fragments and from real life practice implementations of (F/L)OSS programs. The result is that the "procurement process" has now found a logical place in the story. And that two strategies emerged to implement (F/L)OSS in an organization. And finally that the remaining design principles give guidelines that will enable solutions to be created to help create an environment in an organization in which the adoption of (F/L)OSS is easier.

Then one more question remains to be answered and that is the question on the front of the cover. How to buy something that is free? As was shown in this master thesis the answer to that question, lies for a part in the answer another question: why buy something that is free?

> "Open source provides greater control and comes with greater responsibility. If this is not of value to an IT department, open source will seem like a lot of work" (Woods & Guliani, 2005).

6 Limitations, future research and reflection

In this chapter the limitations of the research are presented, suggestions for further research are made and a reflection is given on the followed methodology of the research.

6.1 Limitations

The results and implications of this study should be interpreted in terms of it's limitations. First the field that was investigated is relatively new and apart from the adoption framework, no theoretical framework existed to guide the research, or that the research could test. Therefore this research has an exploratory nature.

Second the generalizability of the research is limited. Only six cases were investigated and results are bound to these cases. However there are indications that the research can be generalized beyond these six. These are formed by forum reports (Slabman, 2010), news articles (Gelles & Waters, 2010; Modine, 2009; Heur, 2010; Hulsman, 2010) and existing description of cases in literature (Cassell, 2008; Hauge et al., 2008; Glynn et al., 2005) of people in similar conditions. These people mention mechanisms and interventions very much in line with this research. This gives some indication that the results can be applied broader.

Another argument that the research can be generalized beyond the cases is that the culture of the people dealing with the issues mentioned in this research is broader. It can be said that at least some of the people working in IT-departments feel part of the hacker-culture. And people that a part of this culture are found across industry sectors. This shows that although the research took place in a government setting, the actual culture in which the research took place, is widespread. Therefore it could be argued that the results might apply beyond the cases.

A third limitation is that the design principles have only been designed and not tested. Only a review by experts from the field took place. Although this review did not result in many comments, it is not proven that the design principles interventions function correctly and that the mechanisms are precisely understood. Again this makes this research of highly exploratory nature and further research is necessary.

6.2 Further research

Four areas of research are suggested to be investigated further based on the results of this thesis work. The first one is testing these design principles in practice. The second investigating the effect of an IT-architecture policy on the adoption of (F/L)OSS. Thirdly research should be done on the generalizability of the design principles to firms. And last the forming of a (F/L)OSS learning organization should be investigated.

Future research should investigate if these design principles function correctly in the field of prac-

tice. It should confirm if the mechanisms are working the way they are understood and presented in this thesis. Executing this kind of research would complete the research-design cycle (Romme & Endenburg, 2006).

Second the influence on (F/L)OSS adoption of an IT-architecture should be investigated. Especially in relation to the existence of incompatible (F/L)OSS knowledge within the IT-staff. Also research should be done to how to formulate just exceptions within the IT-architecture.

Third can the corporate world make use of these design principles too? Why are they working, or why not. Are there differences in the underlying mechanisms?

The fourth suggestion for further research has to do with the subtle the conflict between paradigms. (F/L)OSS stems from the perspective that a software program is nothing more than highly codified knowledge. Academic scholars are used to discussing and sharing knowledge. A very similar system is found in the (F/L)OSS projects. It can be said that the (F/L)OSS community resembles the academic community in this regard. Other people think that software is nothing more than a product. And that the knowledge contained within needs to be protected by patents and copyright. Can an organization that invests in (F/L)OSS knowledge more efficiently adjust to changes in the (business) environment because of this knowledge sharing process? The goal should be to create a learning environment that lives and breathes (F/L)OSS in order to efficiently implement business process adjustments. This could be of high value because: "The moral? An organization's capacity to improve existing skills and learn new ones is the most defensible competitive advantage of all"(Hamel & Prahalad, 2005).

6.3 Reflection

The work necessary for the execution of this research was extensive. This was largely due to the elaborate coding sessions. But these were necessary. Yin (1984) states that when building explanations "the investigator might slowly drift away from the original topic of interest". It has been tried to prevent this from happening by using an elaborate systematic analysis. To reduce this problem, all interviews were first coded on general pre-defined codes (strains, issues, coped, resolved) and then open coding was used to find emergent themes within the challenge-cope-sets. This was repeated per case. Then all the six cases were again compared on the basis of this coding. Of course these codes did not always align perfectly, so the case interviews had to be revised and the open codes redefined until eventually six main themes were distilled. This iterative process was extremely time consuming but worth the effort. During the analysis it became apparent that looking only at the challenges, ignores the aspects that went well. This was resolved by reviewing the transcriptions of the interviews again during the analysis.

A Detailed descriptions of the processes

Step	Summary
Motive	The undersecretary of Economic affairs was looking for a pilot. DPO wanted to do this, but DGET demanded a business case that would demonstrate the feasibility of an open desktop at DPO.
Business case specifications	Specifications were made for a busines case that would show that an open desktop would be feasible.
Business case selection	The company BartIT was chosen on the basis of price to execute the business case.
Business case execution	The busines case was executed in three weeks and contained four options. The thin client option appeared to be the most appropriate for DPO.
Action plan	After the positive business case, the DPO was mentioned as a pilot project in the action plan "NOiV" of the undersecretary.
Project plan	After the goals were set by the mentioning of DPO in the national action plan "NOiV", DPO started to create a project plan. This plan investigated what had to be dis-entangled and or replaced. This resulted in the forming of four teams and two project phases. For the successful execution of the project plan, external help was necessary. DGET had reserved budget for this. After a normal tendering procedure, ATcomputing was awarded the contract to assist DPO.
"Thinking" phase	In this phase ATcomputing and DPO started planning for how and what (F/L)OSS would be used and what systems of the ICT at DPO needed to be replace. This took about half a year to gather all the possibilities. In this phase the following principle was used: Does the program run on Linux? Yes then install this option. If not, is there an open source alternative? If so use that. If not then distribute the application via terminal service. This approach is application based. Per application the lockin is removed and the new application is only deployed to the users after it is fully implemented.
Implementation	These systems were replaced: CMS, CRM, email and groupware servers, storage servers. For the selection between alternatives of each of these type of applications a long list was created. This list was then filtered on the basis of specific functional wishes. That resulted in a short list of around three alternatives. These were then installed and tested by the end-user. The end-user made the final decision.

Table A.1: Summary of the process followed in the Dutch Patent office case.

Time	Step	Summary
2006	Motive	Within the Dutch Defense organization, a need arose for more Internet access. Internet was provided by a terminal service and this service was reaching it's bought maximum concurrent user limits, because of the increase in need. Next to this it was not performing very well.
	First improvement	First reaction was more terminal server licenses, more hardware and automatic server management instead of the manual method used until then. This however was expensive and only slightly improved the performance.
2007-2008	Research	A (top)manager noticed the increase in requests of expensive terminal server licences. He knew open source might be an alternative and initiated the PODVIS project at the RIC (R&D department of the IT company of the department of defense).
		The requirements for the new environment were already known, since they were the same as the old environment.
		The (F/L)OSS solution from the R&D department developed as an alternative, required different knowledge. The system administrators of the existing Internet service did not have compatible knowledge.
Begin 2008	Decision	In the mean time the policy of the ministry of defense changed and all the employees were given an Internet option. This caused an even larger strain on the Internet service. At this point, it was decided to scale up the (F/L)OSS proof of concept to a beta. This meant an Internet service with a less strict SLA. This had the effect that no other alternatives were looked at. Because the beta (F/L)OSS solution was performing. So at that time two methods to go online were present.
November 2008 - March 2009	Implementation	When the users were positive about the beta a decision was made to put the new service into final production. Not the system administrators of the existing Internet system would take care of it, but a different product group was given the support task. This was done because that group had compatible knowledge about the (F/L)OSS solution.

Table A.2: Summary of the process followed in the DoD case.

Time	Step	Summary
Two years ago.	Motive	The project was started a couple of years ago. It is the second half of the implementation of a defense ***.
		The goal is to ***. So it is a visible project.
		The project should have already been finished. But due to circumstances it was delayed a long time.
		This year it should be finished. Nothing was said about the need for open source software.
2009	Business case	A short business case was made about the costs and benefits of the *** on the ***. It also mentioned that (F/L)OSS should be investigated.
		In the business case a couple of options were drawn.
	Impact analyze	Then the project manager did an impact analysis (more in depth analysis) of the business case. And discussed the scope with the program manager.
		They found out that in order to expeditiously get the system going, they would not opt for a (F/L)OSS solution. Also they needed to integrate with legacy software. Using (F/L)OSS
		would have taken up too much time to research all the technical details and implications.
	Project Proposal	With the information form the impact analysis and the business case documents a project proposal was written. This proposal then went through the DoD financial checks and got approved.
	PID	The project leader now made a project initiation document with a more fine-grained planning and scope. They decided not to do the *** part in this project.
		The *** would just be to *** and not for instance to ***.
2010	implementation	After approval of the PID and the budget had been set the project started. Within a year the new system will be running.

Table A.3: Summary of the process followed in the Z case.

Step	Summary
Reason	The IT-department knew about the national policy "actie plan NOiV" of undersecretary Heemskerk. They also felt the need to do something with it. Next to this they had a large request of Adobe Writer licenses. This was something that they thought was strange. And therefore they started an inquiry.
Research	They found a need for a conversion program that made pdf's out of .doc. This provided an opportunity to do something with open source. An (F/L)OSS program was found and tried out to see if it was technically feasible. They also wanted to become familiar and gain trust in their own capability to support and maintain it.
Selection	The two options were presented to the internal customer. He could choose between paying for the licenses or installing the "free" option that would also solve the problem. The decision was easy. Then the product keeper was asked to get the obtain the product.
Contracting	The product keeper contacted the legal department which read the license of the (F/L)OSS program. Ordering a cd with the program, like how it was done in the research phase, was not the legally correct way. The program should be downloaded.
Obtain	Obtaining the product was a problem. Because downloading was not allowed. And ordering a cd was not possible on legal grounds.
Register	Registering the product was also a problem. The system that tracks all the software did not accept a product that had the price value "0".

Table A.4: Summary of the process followed in the Dutch tax office case.

Tijd	Stap	Samenvatting
2004	A7 Project	Grootegast started four years ago with an investigation into open source software.
		This was done, because the philosophy as expressed by the motion Vendrik, struck a chord with the IT department.
		The A7 report mentioned thin clients and open office, but it was concluded that it was too early to switch.
2008, November	Motive	Then in 2008, a choice had to be made for a new office program. Also things were changing in the network, and the government was pressing open source. The open standard ODF, became mandatory.
November till March	Research	
		An investigation was started to see if open office could now be introduced. The municipality already had experience with open source software.
		In te investigation also the lockin situation was addressed and a way to deploy and maintain open office was found in a non-free program from the US.
2009	Presentation to the management team	All this information from the second and first investigation was gathered.
		A company that would guide the project was hired to show the management team what the plans were and the budget was.
March	Choice	
		The management team was soon convinced also since it had already agreed on policy 4 years ago and because of the recent policy from the Dutch government.
		So all the plans and facts were presented to the "college of B&W", who decided to proceed with the project. Giving it even more legitimacy.
	Deliberate with suppliers	Then talks started with the proprietary software vendors that did not adhere to the open standards. They were pressured into giving support.
		All of them agreed and were already or were planning to work on open standards.
	Pilot	A pilot was then started with 4 people to see what the users would say about the new program. Even with little training they could operate the software comfortably.
November till July	Implementation	Then the software was made ready for the final release in the network. The whole process took about 9 months.
	Training	During the pilot and early introduction, experiments were held with the training. The combined training was not a success. The people preferred "one-to-one" training which was more efficient
2010, January	Evaluation	Today they are preparing for an evaluation of the system and a second deployment for a neighboring community.

Table A.5: Summary of the process followed in the Grootegast case.

Time	Step	Summary
2007	Reason	Their ICT was in need of an update. This upgrade had been postponed, because of an opportunity to work together on ICT with other municipalities. However an agreement could not be reached . So by then the municipality was in desperate need of an ICT update. Next to this they also wanted a new house style.
June 2007	Research	Voerendaal saw an opportunity to do something with (F/L)OSS in the pending changes. They had talked with Vaals, a neighboring municipality that had already introduced OpenOffice.org. Voerendaal was enthusiastic so they called the same company that had helped Vaals. This company explained some general things that Voerendaal needed to consider.
August 2007	Research	A day of training and research for the IT staff, guided by the (F/L)OSS company. At that time Voerendaal had put together a plan of attack and a list of wishes for the new system.
September 2007		The by then officially unsupported Novell servers were replaced by new ones.
	Pilot	A pilot was executed with Office 2007 from Microsoft. The users were unhappy with the Office 2007 interface. And much happier with the OpenOffice.org interface. At this point the "college van B&W" had given formal support for the deployment of OpenOffice.org.
December 2007		The (F/L)OSS company worked on a document generating system, to support the use of the new house style.
January 2008	Second pilot	House style/Document system finished. Start with the OpenOffice.org pilot.
March 2008	Implementation	Some changes and improvements based on the pilot results were made.
April 2008	Implementation	Final implementation in one step including the new pc's. They also gave the employees a usb stick with a portable version of OpenOffice.org.

Table A.6: Summary of the process followed in the Voerendaal case.

B Interview protocol

Interview protocol (naar Miles & Huberman)

Praktisch: Noteer naam, email en telefoon nummer Is er documentatie van de beslissing ? Mag ik contact opnemen met vervolg vragen Wie kan ik nog meer interviewen?

NB: Vragen 1 t/m 5 moeten aan bod komen in het interview. De andere vragen kunnen ook achteraf via mail of bellen.

1) Hoe is het hele proces verlopen? Welke afwegingen zijn er gemaakt? Wat waren belangrijke momenten?

Probes: top down opdracht vs bottom up innovatie proces vs eigen traject/verantwoording Requirements? Wie? Iedereen ermee eens? Zinnig? Vergeleken vs bias voor oplossing? Reden: strategie vs opportunisme (operationele redenen) Waarom deze oplossing? Vendor? (F/L)OSS vs COTS Zijn er eisen gesteld aan vendor? SLA? Was er een vendor selectie proces? Had de vendor invloed op het beslissings proces / beslissing nemer? Wanneer werd de vendor bij het beslissings proces betrokken?

tijd: 25 min

2) Waren er uitdagingen/problemen bij de beslissing? Welke?

Probes: Bedrijfspolitiek Eerlijk besluit? Goed vs fout Hoe voldoet de (F/L)OSS oplossing? (ook weer (F/L)OSS vs COTS) Werkte de requirements? Kreeg men wat men wilde? TCO? ROI?

Tijd 10 min

3) Hoe zou het beslissings proces beter kunnen?

Probes: (F/L)OSS voldoet? COTS? ROI? TCO? Politiek? Strategie?

Tijd 10 min

4) Welke strategische overwegingen/keuze speelde een rol of zijn er gemaakt?

Probes: leveranciersonafhankelijkheid? speelde dit een rol in de beslissing? Wat is uw eigen visie? Is er strategische visie op dit gebied, naast âĂIJbij gelijke geschiktheidâĂİ

Tijd 10 min

Wat voor voordelen heeft (F/L)OSS volgens u?

Probes: verschil tussen (F/L)OSS en COTS? Wat als project stopt? Eisen aan community? Vs bedrijf failliet en SLA?

tijd 5 min

Hoelang duurde het project? Vanaf het punt van signalering van de noodzaak voor de software tot het punt van de eerste release.

Probes tijd beslissing traject: reden voor de duur: limiet ja/nee deadline voor de implementatie ja/nee wie stelde deadline/limiet: /

tijd 5 min

Wie waren er betrokken bij de beslissing en wat was hun rol?

Probes: Uw rol: Wie nog meer + rol Wie verantwoordelijk? Wie nam eind beslissing ja/nee dit systeem/ontwerp Wie kwam met idee voor (F/L)OSS? tijd 5 min

Gaan eigen verbeteringen terug naar (F/L)OSS project?

Probes: Hoe ziet het interne development proces eruit? Project structuur en management? Formeel proces? Kosten/tijd leveren van support? Wie is verantwoordelijk?

tijd 5 min

C Complete synthesis and list of final design principles

In this appendix all the final design principles are shown including their synthesis. They are organized by the themes process, entrepreneurship, top management support, lock-in, architecture, organizational support and change management.

The final design principles are presented in a textbox. In this textbox the practice-based design principles are shown on the left, the final design principle in the middle and the theory-based design principle on the right. By marking text of the practical- and theory-based with a different color, the origin (theory or practice) of the text in the final design principle can be traced. Also note that in the textbox of the final design principle references have been made to show what theory or case(s) are supporting the text.

Acquiring (F/L)OSS

There are two practice-based design principles about how to deal with (F/L)OSS. The first one shows the choice between configuring a (F/L)OSS solution yourself, having other companies do this configuring for the organization, or getting help to compensate the knowledge incompatibility. The last to involve a procurement process while the first one does not.

The second practice-based design principle is about strategies that can be followed when configuring a (F/L)OSS solution yourself. Therefore the synthesis begins with practice-based design principle 1, because the context is broader.

This context however cannot be combined with the theory-based design principle, because the theoretical principle is for (F/L)OSS knowledge and/or integrated solution *procurement* only. The practice-based principle specifies that a strategic choice to *use* (F/L)OSS must have been made. This is different since (F/L)OSS can't be purchased, but knowledge about (F/L)OSS and integration services could be purchased. So the practice based context and thus also the outcome remains the same in the final design principle.

Then the first intervention of the practice-based design principle is "downloading, installing and configuring (F/L)OSS yourself". This is the context of the strategies mentioned in practice-based design principle 2. So the interventions of these strategies should be placed here. To save place only a reference was made. One particular intervention of practice-based design principle 2 deserves special attention and that is the planning. Each time (F/L)OSS is downloaded, installed and configured by an organization, this undertaking should be planned in great detail in order to limit risks.

The next two interventions are about procuring integration services or knowledge services. For each of these a regular procurement process can be followed. This process is described by the theory-based design principle. So the intervention of this principle, can be combined with these practice based interventions.

Practice-based design principle 1

c In the context of a (strategic) choice for (F/L)OSS use

o implementing (F/L)OSS can be done,

i by downloading and installing and configuring (F/L)OSS yourself,

m because when there is enough knowledge of (F/L)OSS, this is possible and (F/L)OSS allows everyone to do this and it is possible to by-pass a procurement department.

i by procuring a (F/L)OSS solution from a system integrator,

m because there might not be enough internal knowledge about (F/L)OSS or the organization wishes to outsource IT then having a system integrator configure the solution is an option.

i by procuring the support of a (F/L)OSS knowledge specialist to assist with the downloading and installing and configuring of the (F/L)OSS solution,

m because there might not be enough internal knowledge about (F/L)OSS but the organization wants to internalize this knowledge in order to become self (reliant) proficient with (F/L)OSS.

Practice Design principle 2

c In the context of replacing a business critical system yourself,

o implementation of (F/L)OSS is achieved,

i by downloading and installing in parallel a (F/L)OSS solution with the same functionality as the operational business critical system and gradually evolving this (F/L)OSS solution until it is mature and proven to replace the current system (DPO,DoD,Grootegast),

m because the evolving approach requires beta releasing which can build organizational support for the solution. It also minimizes risk and can demonstrate that it's functionally sound to replace the system.

c In the context of a non business critical process,

o creating a precedent for the use of (F/L)OSS can be done,

i by downloading and implementing the (F/L)OSS solution yourself (DTO),

m because this method does not require a lot of resources from top management, but entrepreneurial spirit of the implementer will find an opportunity to use (F/L)OSS.

c In the context of introducing completely new ICT functionality yourself,

o implementation of (F/L)OSS is achieved,

i by downloading, installing, experimenting with the (F/L)OSS solution and finally lobbying for organizational support or implementing some change management (Voerendaal),

m because the functionality does not exist yet. So experimenting can be done without risk until the solution works. Then to gather organizational support to make adoption of the solution easy, lobbying and change management is required.

c In the context of downloading and implementing (F/L)OSS yourself,

o securing project success is done,

i by planning the project extensively (all cases),

m because there are a lot of factors to consider. What those factors are depend on the specific situation. Time should be taken to investigate what must be changed and when. That way guestimates about technical issues, issues with interconnections and scope and necessary resources (time and money) can be determined for the project.

Final design principle 1

c In the context of a strategic choice for (F/L)OSS use

o implementing (F/L)OSS can be done,

i by downloading and installing and configuring (F/L)OSS yourself,

i By using the interventions in the applicable context from practice-based design principle 2

m because when there is enough knowledge of (F/L)OSS, this is possible and (F/L)OSS allows anyone to do this and it is possible to by-pass a procurement department.

i by procuring a (F/L)OSS solution from a system integrator,

i by using a procurement process that uses specific terminology in the requirements to specify the (F/L)OSS characteristics,

m because governments should specify neutral requirements and this terminology will make sure the (F/L)OSS characteristics are translated to valid neutral requirement statements that will request (require) (F/L)OSS without asking for it specifically.

m because there might not be enough internal knowledge about (F/L)OSS or the organization does wishes to outsource IT then having a system integrator configure the solution is an option.

i by procuring the support of a (F/L)OSS knowledge specialist to assist with the downloading and installing and configuring of the (F/L)OSS solution,

i by using a procurement process that uses specific terminology in the requirements to specify the (F/L)OSS characteristics,

m because governments should specify neutral requirements and this terminology will make sure the (F/L)OSS characteristics are translated to valid neutral requirement statements that will request (require) (F/L)OSS without asking for it specifically.

m because there might not be enough internal knowledge about (F/L)OSS but the organization wants to internalize this knowledge in order to become self (reliant) proficient with (F/L)OSS.

Theory-based design principle 11

c In the context of (F/L)OSS services procurement,

o (F/L)OSS knowledge or a custom build integrated (F/L)OSS solution can be obtained,

i by using a procurement process that uses specific terminology in the requirements to specify the (F/L)OSS characteristics,

m because governments should specify neutral requirements and this terminology will make sure the (F/L)OSS characteristics are translated to valid neutral requirement statements that will request (require) (F/L)OSS without asking for it specifically (Ghosh, 2010).

Table C.1: Acquiring (F/L)OSS

Entrepreneurship

First looking at the context, the practice-based design principle does not mention adoption. This was added since from the theory it was learned that downloading is not procurement and that here some of the adoption factors are driving the acquiring of (F/L)OSS. So it's warranted to call this adoption of (F/L)OSS to reflect this. The question is how much the qualification "organizational" and "environmental" factors would add to the final design principle. It can be argued that "yourself" includes the organization and "downloading" involves interaction with the environment. So these qualifications are left out of the final principle.

Comparing the outcome, it is learned from the theory that it's actually individual entrepreneurialism. This can also be seen in the cases. Were there is one champion that started the (F/L)OSS endeavor (DTO, DPO, Voerendaal and Grootegast). So this "individual" is added to the final principle.

To gain the the individual entrepreneurialism one must state a precise definition of a strategic goal. This can be made more specific with the next theoretical intervention "issuing strategic intent". This is compatible because both involve stating a goal. However including strategic intent, appears to have personal effort, employee commitment and triggering adoption as outcome. These outcomes can therefore be added to the final outcome. Looking at the practice-based design principle, there is actually a compatible intervention, namely pointing out national policy. National policy can be considered a strategic goal because the employees are in service of the government that issued the policy. So in the specific case of (F/L)OSS adoption by the government, the national policy can be seen as an issue of strategic intent.

The second theoretical intervention in order to create individual entrepreneurialism is "organizing a high perceived capability for solutions". This is compatible with the mechanism of the second intervention of the practice-based design principle. Experimenting with (F/L)OSS, will give the employees experience and they gain knowledge and self confidence in handling (F/L)OSS. This thus creates a high perceived capability for ((F/L)OSS) solutions.

High perceived capability for ((F/L)OSS) solutions can also be gained by knowing about possible (F/L)OSS solutions. Therefore it is necessary to be aware of the different (F/L)OSS programs. From the practice-based design principle it is learned that this is done by educating employees in different kinds of (F/L)OSS alternatives. From the theory it is learned how to execute this "educating" more specifically. By "monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines".

The mechanisms of the interventions just follow from the coupling. They cannot be altered since they explain why an intervention works. So they must align and or complement each other, otherwise more explaining is necessary. From reading the mechanisms of the final design principle it can be seen that the mechanisms complement each other.

Top management support

First the contexts of the design principles are compared to see if the contexts can be combined. The practice-based design principle 4 has the context of acquiring (F/L)OSS in a government setting which is a board context. This can be fitted with tbdp 2, because in acquiring (F/L)OSS decisions must be made about IT ((F/L)OSS is IT). Before obtaining top management support managers need to decide rationally otherwise the convincing won't work. So first the perceptions need to change. This is for instance done by pointing out the national policy on (F/L)OSS. Therefore the practical context must be refined because changing perceptions is only necessary in complex IT decisions with incomplete information. Then these decisions are made on the basis of arguments. One of

Practice-based design principle 3

c In the context of downloading and implementing (F/L)OSS yourself,

o stimulating the entrepreneurial attitude towards (F/L)OSS of employees, can be done,

i by pointing out the national policy and the (F/L)OSS philosophy (all cases),

m because this serves as a trigger to start searching for opportunities to use (F/L)OSS.

i by giving employees room for experimentation (DPO, DoD, Grootegast, Voerendaal),

m because this will make it easier to determine if a (F/L)OSS solution meets the requirements.

i by educating employees in different kinds of (F/L)OSS alternatives (DPO) ,

m because this will increase the chance that opportunities are identified to use (F/L)OSS.

Final design principle 2

c In the context of adoption of (F/L)OSS by downloading and implementing (F/L)OSS yourself,

o stimulating the individual entrepreneurial attitude towards (F/L)OSS of employees and obtaining personal effort, employee commitment and triggering adoption can be done,

i by stating a precise definition of a strategic goal,

- i by issuing strategic intent (a clear in goal) but being flexible in means to reach this goal,
  - i by pointing out the national policy and the (F/L)OSS philosophy,
  - m because this serves as a trigger to start searching for opportunities to use (F/L)OSS.
- m because current resources and capabilities will not suffice and thus the organization is forced to be more inventive and make the most of the limit resources.

m because this will lead to an understanding of the challenge and helps identifying an opportunity.

i by organizing a high perceived capability for solutions.

- i by giving employees room for experimentation,
- m because this will make it easier to determine if a (F/L)OSS solution meets the requirements.
- i by educating employees in different kinds of (F/L)OSS alternatives,
  - i by monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines,
  - m because all these methods will bring external current (F/L)OSS program knowledge into the organization and thus make people aware of the existence of (F/L)OSS.
- m because this will increase the chance that opportunities are identified to use (F/L)OSS.

m because knowing what the team is capable of doing will also identify what the possibilities for a solution are and what capabilities must be acquired in order to implement the solution.

Theory-based design principle 7

c In the context of organizational factors influencing the adoption of (F/L)OSS,

o individual entrepreneurialism is created,

i by stating a precise definition of a (urgent?) strategic goal (Stopford & Baden–Fuller, 1994),

m because this will lead to an understanding of the (present) challenge and (Stopford & Baden-Fuller, 1994) helps identifying opportunities for solutions (Hamel & Prahalad, 2005).

i by organizing a high perceived capability for solutions

m because knowing what the team is capable of doing will also identify what the possibilities for a solution are and what capabilities must be acquired in order to implement the solution (Stopford & Baden-Fuller, 1994).

Theory-based Design principle 5

c In the context of organizational factors influencing the adoption of (F/L)OSS,

o obtaining personal effort, employee commitment and triggering adoption, is done by

i by issuing strategic intent (a clear in goal) but being flexible in means to reach this goal (Hamel & Prahalad, 2005),

m because current resources and capabilities will not suffice and thus the organization is forced to be more inventive and make the most of the limit resources (Hamel & Prahalad, 2005).

Theory-based design principle 8 (part)

c In the context of external environmental factors influencing (F/L)OSS adoption,

o creating more awareness about (F/L)OSS can be done,

i by monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines (Nagy et al., 2010) ,

m because all these methods will bring external current (F/L)OSS program knowledge into the organization and thus make people aware of the existence of (F/L)OSS. (Nagy et al., 2010)

Table C.2: Entrepreneurship

these arguments is found in tbdp 1 can be integrated in the intervention of pbdp 4. And finally an answer to why top management support is needed is given by tbdp 6 and this would adjust the context with radical high risk initiatives. Something that (F/L)OSS can be, depending for instance on the amount of existing skill in the organization.

Then the interventions can be folded together. Top management support is generated in practice by convincing top management with reasons (pbdp 4). A successful business case or proof of principle will prove that the solution is technically viable. Costs are also a reason (intervention pbdp 4). This intervention can be combined with tbdp 1. Offsetting the true cost of proprietary software against (F/L)OSS provides arguments to create top management support. Because if by opting for a proprietary alternative vendor lock-in is created, this will limit the choice when new investments in software are made. The costs incurred moving away in the future from this proprietary software should be added to the current costs of the proprietary product.

lock-in

In the description of the practice-based design principle the term vendor lock-in is used. The theoretical equivalent is technical compatibility. Actually lack of compatibility. So these to contexts can be integrated. The outcomes are the same; reduce lock-in.

Then the first practice-based intervention, does not have a theoretical counter part so it is just copied to the final design principle. Then the second practice-based intervention is the same as the theory-based intervention. Looking at the mechanisms of both interventions, the practice-based mechanism has additional explanations, how the (F/L)OSS characteristics make vendor lock-in impossible. So these mechanisms are added.

The rest of the interventions and mechanisms of both practice and theory are additional interventions how to reduce lock-in.

Architecture

The context of the practice based and theory based principles, can be synthesized...

The interventions of the theory and the practice not completely the same. It appears that only having an IT architecture is not enough. It is also necessary to let the policy be guarded by top management and supporting processess in the organization.

In that case the outcomes can be combined. The theory adds the protection of previous (F/L)OSS investments to the “strategic application of ICT”.

Finally the mechanisms are combined. They already were the same. The practice based principle only adds the motivational aspect created by the guarding of the policy. Which is also logical because now employees will be forced to take (F/L)OSS into serious consideration. But this time not on the basis of operational aspects, but on the basis of the strategic goals set forth by top management.

Organizational support

The practice-based design principle contains four interventions to achieve the outcome “creating organizational support”. The first intervention is about “explaining” and “showing what (F/L)OSS programs exist”. This can be compared with the outcome of the first theory-based design principle.

Practice-based design principle 4

c In the context of acquiring (F/L)OSS in a government setting,

o obtaining top management support is done,

i by convincing top management with reasons (compliance with national policy, proof, costs calculations) (all cases),

m because compliance with national policy and a successful business case shows confidence that the solution can succeed. So a risk avoiding manager is more inclined to agree. And lower costs are usually something managers aspire to by default, especially in government settings since it is public money that is being spent.

Final design principle 3

c In the context of an radical, high-risk initiatives such as acquiring of complex FLOS-software in a government setting with incomplete information about the decision,

o obtaining necessary resources such as time, space, equipment, people and communicating, supporting and sharing, vision and goals is done,

i) by creating top management support,

i by changing the perceptions that managers have of (F/L)OSS and providing them with information about (F/L)OSS,

i by convincing top management with reasons of compliance, proof of principle and/or business case,

m because compliance with national policy and a successful business case shows confidence that the solution can succeed. So a risk avoiding manager is more inclined to agree.

i by convincing top management with reasons of costs calculations

i by calculating the switching cost: what it would cost to implement the proprietary software now and change back to (F/L)OSS later,

m because these costs belong to the proprietary systems since they would not be incurred when adopting for (F/L)OSS now.

m because lower costs are usually something managers aspire to by default, especially in government settings since it is public money that is being spent.

m because this will prevent informational cascading and reduce reputation herding. Knowledge is gained and thus the urge to imitate the decisions of other managers is reduced. Furthermore taking away irrational fear of damage to the managers reputation will cause them to decide more rational.

m because these resources are vital to the success of such an initiative and only top management can provide these resources because they have the legitimacy and power to distribute them.

Theory-based design principle 9

c In the context of complex IT decisions with incomplete information,

o a managers choice based on rationality and in the best interest of the organization is created,

i by changing the perceptions that managers have of (F/L)OSS and providing them with information about (F/L)OSS,

m because this will prevent informational cascading and reduce reputation herding. Knowledge is gained and thus the urge to imitate the decisions of other managers is reduced. Furthermore taking away irrational fear of damage to the managers reputation will cause them to decide more rational (Miralles et al., 2005; Morgan & Finnegan, 2007).

Theory-based design principle 1

c In the context of formulating arguments to support adoption of (F/L)OSS,

o determining the cost for a (radical) (F/L)OSS implementation can be done,

i by calculating the switching cost what it would cost to implement the proprietary software now and change to (F/L)OSS later,

m because these costs belong to the proprietary systems since they would not be incurred when adopting for (F/L)OSS now (Ghosh, 2010).

Theory-based design principle 6

c In the context of radical, high-risk initiatives and absence of slack resources,

o obtaining necessary resources such as time, space, equipment, people and communicating, supporting and sharing, vision and goals is done:

i by creating top management support,

m because these resources are vital to the success of such an initiative and only top management can provide these resource because they have the legitimacy and power to distribute them (Glynn et al., 2005; Masrek et al., 2009).

Table C.3: Top management Support

Practice-based design principle 5

C In the context of vendor lock-in caused by legacy technology,

o reducing lock-in is done,

i by convincing the existing vendors to use open standards (Voerendaal, Grootegast),

m because open standards make it possible to interchange parts of a total IT solution.

i by replacing as much software that causes lock-in with (F/L)OSS (DPO),

m because (F/L)OSS uses open standards and thus makes it possible to interchange parts. And by using (F/L)OSS one does not rely on a specific vendor but instead can get support from many different companies.

i by preventing that current investments will make future investments of (F/L)OSS more difficult(Case Z),

m because lock-in will at least not be increased by the new investments. This will also prevent these current investments to become sunken costs in the event that (F/L)OSS is chosen in the future.

Final design principle 4

c In the context of technical compatibility issues (vendor lock-in) with legacy technology,

o reducing the lock-in is done,

i by convincing the existing vendors to use open standards,

m because open standards make it possible to interchange parts of a total IT solution.

i by replacing the legacy systems by installing (F/L)OSS (alternatives) as much as possible,

m because the characteristics of (F/L)OSS make vendor lock-in impossible. (F/L)OSS uses open standards and thus makes it possible to interchange parts. And by using (F/L)OSS one does not rely on a specific vendor but instead can get support from many different companies.

i by encapsulating the parts that cannot be replaced, by using middleware and virtualisation techniques,

m because these will make it possible to limit the amount of the legacy software and thus increasing the opportunity to use (F/L)OSS.

i by preventing that current investments will make future investments of (F/L)OSS more difficult,

m because lock-in will at least not be increased by the new investments. This will also prevent these current investments to become sunken costs in the event that (F/L)OSS is chosen in the future.

Theory-based design principle 2

c In the context of technical compatibility issues with legacy technology,

o to reduce lock-in,

i by replacing the legacy systems by installing (F/L)OSS (alternatives) as much as possible (Nagy et al., 2010),

m because the characteristics of (F/L)OSS make vendor lock-in impossible.

i by encapsulating the parts that cannot be replaced, by using middleware and virtualisation techniques,

m because these will make it possible to limit the amount of the legacy software and thus increasing the opportunity to use (F/L)OSS(Nagy et al., 2010).

Table C.4: lock-in

Practice-based design principle 6	Final design principle 5	Theory-based design principle 10
c In the context of acquiring (F/L)OSS,	c In the context of acquiring (F/L)OSS,	c In the context of acquiring (F/L)OSS
o implementing an architectural IT policy will structure the strategic application of ICT,	o implementing an architectural IT policy will structure the strategic application of ICT and protecting (F/L)OSS investments can be done,	o protecting (F/L)OSS investments can be done by,
i by creating an architectural ICT policy guarded by top management and supporting processes in the organization(DPO,Grootegast,case Z),	i by creating an IT architecture policy based on (F/L)OSS and open standards guarded by top management and supporting processes in the organization	i creating an IT architecture based on (F/L)OSS and open standards(Feeny & Willcocks, 1998; Ghosh, 2010),
m because this will put a price on non compliance to the architecture and thus will make sure employees have the necessary motivation to create room in projects or procurement's and thus choose technology that is strategically beneficial to the organization.	m because defining the basic principles of (F/L)OSS that are in line with the strategy of the organization in a formal policy at a high level in the organization will put a price on non compliance to the architecture and thus will make sure employees have the necessary motivation to create room in projects or procurement's and thus choose technology that is strategically beneficial to the organization and will make it difficult to argue for proprietary systems that are not in the best strategic interest of the organization.	m because defining the basic principles of (F/L)OSS that are in line with the strategy of the organization in a formal policy at a high level in the organization will make it difficult to argue for proprietary systems that are not in the best strategic interest of the organization.

Table C.5: Architecture

Because it's outcome "creating more awareness" is also the goal of the intervention of the practice-based design principle. So the theory-based intervention gives a more detailed intervention how to create awareness. That is why these are combined.

Then the second practice-based intervention, is a more detailed description of the first intervention of the second theory-based design principle. Because increasing trialability is done in practice by setting up pilots. So these two interventions can be combined.

The third intervention is about communicating which is explained in more detail by the second intervention of the second theory-based design principle namely "showing employees the benefits and communicating experiences and results".

The final practice based intervention can be combined with the intervention found in the third theory-based design principle. Basically they say the same, only the theory-based mechanism adds more explanation about the usability gain in relation to training.

All these interventions will increase the organizational support and the theory-based design principles supported the practiced-based principles by making more clear why the intervention would work.

Change management

No matching theory-based design principles were found. So the final design principle is the same as the practice-based design principle.

Practice-based design principle 7

c In the context of downloading and implementing (F/L)OSS yourself,

o creating organizational support can be done,

i by reducing misunderstanding by explaining what (F/L)OSS is and informing employees about the possibilities (DoD,DPO)

m because showing what exists will make employees understand and familiar with (F/L)OSS

i by making use of the (F/L)OSS characteristic that easily a fully working pilot setup can be installed one can let the end-users of the organization in on the final IT decision(DPO),

m because letting end-users in on the decision of the final solution will generate the feeling that they took part in the project which will create internal support for the final solution.

i by communicating continuously with your end-users(Grootegast,DPO),

m because this can explain the situation and goals and will thus reduce the fear for the solution.

i by training end-users in the new technology(Voerendaal,Grootegast),

m because this will show that the solution works and thus reduce the uncertainty and doubt about the new technology.

Final design principle 6

c In the context of downloading and implementing (F/L)OSS yourself,

o creating organizational support can be done,

i by reducing misunderstanding by explaining what (F/L)OSS is and informing employees about the possibilities

- i monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines,
- m because all these methods will bring external current (F/L)OSS program knowledge into the organization and thus make people aware of the existence of (F/L)OSS.

m because showing what exists will make employees understand and familiar with (F/L)OSS

i by increasing trialability, letting end-user play with the solution

- i by making use of the (F/L)OSS characteristic that easily a fully working pilot setup can be installed one can let the end-users of the organization in on the final IT decision,
- m because letting end-users decide the final solution will generate the feeling that the took part in the project which will create internal support for the final solution.

m because having end-users experience the solution will enable the possibility to test if the solution fulfills the requirements and will give the end-user a feeling that their wishes are being heard.

i by communicating continuously with your end-users,

- i by increasing observability, show employees that the solution creates benefits for them by communicating experiences and results
- m because the more visible the results of technology the more likely the innovation will be quickly adopted.

m because this can explain the situation and goals and will thus reduce the fear for the solution.

i by training end-users in the new technology,

- i by training employees in the technology
- m because training will increase the usability of the (F/L)OSS software and thus the usefulness.

m because this will show that the solution works and thus reduce the uncertainty and doubt about the new technology.

Theory-based design principle 8

c In the context of external environmental factors influencing (F/L)OSS adoption,

o creating more awareness about (F/L)OSS programs can be done

i by monitoring (F/L)OSS repositories, select new personal on the basis of knowledge of (F/L)OSS, send key users to conferences and obtain subscriptions to trade magazines (Nagy et al., 2010),

m because all these methods will bring external current (F/L)OSS program knowledge into the organization and thus make people aware of the existence of (F/L)OSS.

o reducing uncertainty about (F/L)OSS quality,

i by using a (F/L)OSS maturity model (Aberdour, 2007; Zhao & Elbaum, 2003),

m because this will provide an objective measure of the quality of a (F/L)OSS project and thus help assess if the (F/L)OSS program is fit for purpose.

Theory-based design principle 4

c in the context of technology factors influencing adopting (F/L)OSS

o creating support from end-user can be achieved

i by increasing trialability, letting end-user play with the solution (Dedrick & West, 2004; Fichman & Kemerer, 1993),

m because having end-users experience the solution will enable the possibility to test if the solution fulfills the requirements and will give the end-user a feeling that their wishes are being heard.

i by increasing observability, show (training, pilots) employees that the solution creates benefits for them by communicating experiences and results (Xia & Lee, 2000; Venkatesh, 1999),

m because the more visible the results of technology the more likely the innovation will be quickly adopted.

Theory-based design principle 3

c In the context of technology factors influencing the adoption of (F/L)OSS,

o create compatible knowledge and match the level of skill required for the use of (F/L)OSS,

i by training employees in the technology (Zuliani, 2004; Y. Lee et al., n.d.),

m because training will increase the usability of the (F/L)OSS software and thus the usefulness.

Table C.6: Organizational support

Practice-based design principle 8	Final design principle 7	Theory Design principle
c In the context of acquiring (F/L)OSS	c In the context of acquiring (F/L)OSS	c In the context of . . . ,
o creating an environment ready for (F/L)OSS adoption is done:	o creating an environment ready for (F/L)OSS adoption is done:	o . . . ,
i by adjusting work habits(DPO),	i by adjusting work habits	i by . . .
m because the new system might require different ways of performing tasks or offers more efficient ways of performing these tasks, adjustment of work habits might be necessary.	m because the new system might require different ways of performing tasks or offers more efficient ways of performing these tasks, adjustment of work habits might be necessary.	m because . . .
i by adjusting procedures to accommodate (F/L)OSS procurement(DTO),	i by adjusting procedures to accommodate (F/L)OSS procurement	
m because current procedures might not be compatible with certain characteristics of (F/L)OSS. Therefore adjusting the procedures by taking into account these characteristics will help (F/L)OSS procurement.	m because current procedures might not be compatible with certain characteristics of (F/L)OSS. Therefore adjusting the procedures by taking into account these characteristics will help (F/L)OSS procurement.	
i by adjusting existing roles(case Z),	i by adjusting existing roles	
m because the tasks performed in the roles can be the opposite of what tasks are required to introduce (F/L)OSS. Changing the tasks of a role with (F/L)OSS procurement in mind will make sure people are free within their roles to procure (F/L)OSS.	m because the tasks performed in the roles can be the opposite of what tasks are required to introduce (F/L)OSS. Changing the tasks of a role with (F/L)OSS procurement in mind will make sure people are free within their roles to procure (F/L)OSS.	

Table C.7: Change management

D Themes inducement

In this appendix an elaborate commentary is given on the theme inducement table. Per case evidence is presented for the words (yes, no, unknown, limited) without a number and are thus without a corresponding challenge-solutions set. Also per case the column "not classified" is explained. The reason for some of the words is that there is a challenge with the used methodology is to capture actions with a positive effect.

DPO

Because the DPO case was mandated from the under-secretary of economic affairs, it was not a case where entrepreneurial attitude was seen. This of course also caused that there was no problem with top management, because they were the ones starting the project. No challenges were found because of this situation and thus it is difficult to capture in the table.

Challenge-solution set 7 deals with an argument that it's very hard to explain to some managers why (F/L)OSS should be used, if it costs more trouble to make it do the same as proprietary software. This was first classified under "reason why". But in the cross-case it could not be fitted with the other themes. And because it was only found in this case, it was set a side.

DOD

The term limited means that, lock-in was present in the form of the knowledge that the system administrators had, but because the solution was build next to the proprietary application and could be integrated in the existing infrastructure it was decided to call this lock-in limited. The architecture is limited because there is an architecture policy in place, but it's is not mandatory.

Case	Entrepreneurship	Top management support	Lock-in	Architecture	Organizational support	Change management	Not classified
DPO	No	Yes	Yes (5,6)	Yes (8)	Yes (1,3,4)	Yes (2)	7
DOD	Yes (4)	Yes (1)	Limited	Limited	Yes (3,2)	Yes (5)	-
Case Z	Yes (1,5)	Yes (3,4)	Yes (6)	Yes (8)	Unknown	Yes (2)	7
DTO	Yes	Yes (4)	Limited	Limited	Yes (5,3)	Yes (1,2)	6
Grootegast	Yes (5,8)	Yes (7)	Yes (6)	Yes (9)	Yes (1,3,4)	Yes (2)	-
Voerendaal	Yes (1)	Yes (4,3)	Yes (2)	No	Yes (5)	No	6

Table D.1: Inducing themes from cases.

Case Z

If organization support is present in the case is unknown. This simply was not mentioned in the interviews.

DTO

lock-in was limited because the (F/L)OSS software could integrate and was a drop-in replacement for the used proprietary software. Furthermore an architectural policy about (F/L)OSS was in the making. Challenge-solution set 6, was difficult to place in the themes because it was used in the process comparison. It shows that the characteristics of (F/L)OSS change the way one can look at requirements and how organizations could look at software.

Voerendaal

There was simply no architecture policy in place and employees did not have to change their tasks. This was no challenge so it was hard to capture. To make the overview complete, the term no was entered in the table. Challenge-solution set 6 was actually an argument used in the project planning method. Furthermore it did not fit with any other themes. Thus it was set aside.

E Overview of known (F/L)OSS cases

Overview of known (F/L)OSS cases in the Netherlands. The table is a combination of data gathered during this research and a report from Klomparends (2006).

This is just what was known during the investigation. However a look at the NOiV website, tells that there are around 50 municipalities, working on introducing (F/L)OSS. Not all are up and running yet. So an educate guess backed up by Arjen Kamphuis is that there around 50 serious end-user oriented (F/L)OSS implementations currently in the Netherlands. Based on this six cases investigated here, present around 12% of the dutch government end-user oriented (F/L)OSS cases.

Organization	Type	(F/L)OSS program	Date
Den Haag	infra	FreeBSD, Apache, Tomcat, MySQL, Perl, PHP, MMBase	22-01-04
Vlieland	desktop	OpenOffce.org	18-02-04
Haarlem	desktop	XML, OpenOffice.org	03-03-04
Haren	infra	PostgreSQL	16-03-04
Woerden	infra	Linux, Samba, Squid, Apache, PHP, MySQL, OpenOffice.org	30-06-04
A7[1]	infra + desktop	Linux, OpenOffice.org	31-08-04
IJsselstein	infra	PostNuke, Squid, ASSP, OpenOffice.org, JBoss, OTRS, Linux, Apache, MySQL, PostgreSQL, PHP,	01-09-04
Terneuzen	infra + desktop	Linux, OpenBSD, Squid, Packet Filter, Iptables, Dansguardian, OpenSwan, OpenVPN, Sendmail, Mailscanner, Spamassassin, Apache, MySQL, PHP, Samba, OpenOffice.org (pilot in 2005), Nagios, NTOP, MRTG	27-01-05
Almere	infra	Java (J2EE), Linux, PostgreSQL and more.	10-05-05
Vaals	desktop	OpenOffice.org	01-01-03
Voerendaal	desktop	OpenOffice.org	01-06-07
Heerenveen	desktop	OpenOffice.org + diverse other systems	01-01-06
Amsterdam	desktop	Suse Linux Enterprise Desktop, OpenOffice.org, Firefox, Zimbra, Gosa (LiMux)	21-12-09
Dutch tax office	desktop and server	Development tools, pdfCreator, Linux managed platform	02-01-10
Department of defense	desktop and server	Firefox, FreeBSD, Linux	01-06-09
Department of justice	infra	Development tools	01-06-09

Table E.1: Know (F/L)OSS use in the Dutch government.

F Alternative explanations

One of the alternative explanations is given by questioning the interest of proprietary vendors. Some of these local vendors have a vested interest to deliver excellent functionality and provide high levels of support to government organizations, because there livelihood is very much dependant on the satisfaction of the government customer. This also means they will have incentives to listen to wishes of the government in regard to transparency, efficiency and durability. Agreements could be made for instance to have the source code of the proprietary vendor inspected by external trusted parties to ensure transparency. This however would not ensure durability or efficiency. An alternative hinder to the adoption is formed by the knowledge of the society. Since *de facto* "standard" knowledge is largely based on proprietary software products, people from society have relatively little compatible knowledge of (F/L)OSS. And these people could be employees in the organization.

G Interrater reliability

Interrater reliability was calculated by making use of Fleiss' Kappa (Fleiss et al., 1971). This is "a statistical measure for assessing the reliability of agreement between a fixed number of raters when assigning categorical ratings to a number of items or classifying items" (Wikipeadia, n.d.). And the University of Pittsburg has created a webapplication (http://cat.ucsur.pitt.edu), that enables the users to upload a dataset and have raters code the dataset. Then it allows the user to calculate Fleiss' Kappa.

Code	rater a	rater b	rater c	rater d	Exact Match	Partial Match	Kappa
coped	14	48	14	23	4	19	0.50
issues	30	42	43	35	4	45	0.46
process	52	10	28	32	2	33	0.47
strain	16	18	16	8	1	13	0.49
Totals	112	118	101	98	11	110	**0.48**

Table G.1: Interrater reliability results

Landis and Koch (1977) gave the following table for interpreting kappa values.

kappa	Interpretation
‹ 0	Poor agreement
0.0 - 0.20	Slight agreement
0.21 - 0.40	Fair agreement
0.41 - 0.60	Moderate agreement
0.61 - 0.80	Substantial agreement
0.81 - 1.00	Almost perfect agreement

Table G.2: Interpreting kappa values

Bibliography

Abbott, R. (1986). *An integrated approach to software development.* John Wiley & Sons, Inc. New York, NY, USA.

Aberdour, M. (2007). Achieving quality in open source software. *IEEE software*, 58–64.

Ajila, S. A., Wu, D. (2007). Empirical study of the effects of open source adoption on software development economies. *The journal of Systems and Software*, *80*, 1517-1529.

ANP. (2009, March). *Geld voor vrije software.* www.nu.nl.

Applewhite, B. (2003). Should governments go open source? *IEEE SOFTWARE ENGINEERING*, *20*(4), 88-91.

Baetjer, H. (1998). *Software as capital: an economic perspective on software engineering.* (C. Bates, Ed.). IEEE Computer Society.

Benbaset, I., Glodstein, D. K., Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly*, *11*, 369–386.

Boehm, B. (1981). *Software engineering economics.* Prentice-Hall Englewood Cliffs (NJ).

Bonaccorsi, A., Rossi, C. (2003). Why open source software can succeed. *Research Policy*, *32*(7), 1243 - 1258. (Open Source Software Development)

Brooks, F. P. (1995). *The mythical man-month : essays on software engineering* (addison, Ed.). Addison-Wesley Professional.

Brouwer, P., Brekelmans, M., Nieuwenhuis, L., Simons, R. (2010). *Fostering teacher community development* (Tech. Rep.). American Educational Research Association.

BSA. (n.d.). *Don't risk your business - how to ensure your software is licensed.* online, www.bsa.org.

Burg, E. van, Romme, A. G. L., Gilsing, V. A., Reymen, I. M. M. J. (2008b). Creating university spin-offs: A science-based design perspective. *Journal of Product Innovation Management*, *25*(2), 114-128. (Eindhoven University of Technology, Netherlands; Tilburg University, Netherlands)

Burg, E. van, Romme, G. L., Gilsing, V. A., Reymen, I. M. (2008a). Creating university spin-offs: A science-based design perspective. *The Journal of Product Innovation Management*, *25*, 114-128.

Carbone, P., Architect-Nortel, C. (2007). *Value Derived from Open Source is a Function of Maturity Levels.*

Cassell, M. (2008). Why governments innovate: Adoption and implementation of open source software by four european cities. *International public management Journal*, *11*, 193-213.

Centra, C. D. (2006). *Beleidsadviesrapport: Reduceren van leveranciers-afhankelijkheid en de rol van open source en open standaarden.* (Tech. Rep.). Ministerie van Defensie.

Clarke, G. (2009a, May). *That time is gone.* www.theregister.co.uk.

Clarke, G. (2009b, October). *Us dod snuffs open-source 'mis-conceptions' - your secrets are safe.* online. (url-http://www.theregister.co.uk/2009/10/27/department_defense_free_open_source)

Coar, K. (2006). *The open source definition* (Tech. Rep.). The open source iniative. (http://www.opensource.org/docs/osd)

Cohen, H., Keller, S., Streeter, D. (1979). The transfer of technology from research to development.

Research Management, *22*(3), 11–17.

Comino, S., Manenti, F. (2005). Government policies supporting open source software for the mass market. *Review of Industrial Organization*, *26*(2), 217–240.

Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, *49*(6), 564 - 575. (Qualitative Software Engineering Research)

Daffara, C. (2008). *Free/libre open source software: a guide for sme's* (Tech. Rep.). (F/L)OSS metrics and OpenTTT EU projects.

Dedrick, J., West, J. (2004). An exploratory study into open source platform adoption. *IEEE transactions on software engineering*, *8*, 1530-1605. (Los Alamitos, CA, USA)

De Jager, S. (2009). *Corporate venture transition processes at high-tech established firms : the development of design principles for aligning newstream and mainstream business development.* Unpublished master's thesis, Technische Universiteit Eindhoven, the Netherlands.

Denyer, D., Tranfield, D., Aken, J. van. (2008). Developing design propositions through reseach synthesis. *Organization Studies*, *29*(3), 393.

Depietro, R., Wiarda, E., Fleischer, M. (1990). The context for change: Organization, technology and environment. *The Processes of Technological Innovation, Lexington Books, Lexington, MA*, 151–75.

Dinh-Trong, T., Bieman, J. M. (2004). *Open source software development: A case study of freebsd.* (Paper presented at the 10th International Symposium on Software Metrics, Como, IT.)

Dirven, P. (2009, July). *Open source draagt niet bij aan transparantie en kwaliteit.* www.computable.nl.

Eilander, S. (2008). *The acquisition of open source software: A guide for ict buyers in the public and semi-public sectors* (Tech. Rep.). NOiV.

Eisenhardt, K. M. (1989). Building theories from case study research. *The academy of management review*, *14*, 532-550.

Evans, D., Reddy, B. (2003). Government preferences for promoting open-source software: a solution in search of a problem. *Michigan Telecommunications and Technology Law Review*, *9*, 313–457.

Farbey, B., Finkelstein, A. (2001). Software acquisition: a business strategy analysis. *IEEE International Conference on Process Requirements Engineering.*, *1*, 1-76.

Feeny, D., Willcocks, L. (1998). Core IS capabilities for exploiting information technology. *Sloan Management Review*, *39*(3), 9–21.

Ferrin, B., Plank, R. (2002). Total cost of ownership models: An exploratory study. *Journal of Supply chain management*, *38*(3), 18–29.

Fichman, R., Kemerer, C. (1993). Adoption of software engineering process innovations: The case of object orientation. *Sloan Management Review*, *34*, 7–7.

Fitzgerald, B. (2006). The transformation of open source software. *Management Information Systems Quarterly*, *30*(3), 587.

Fitzgerald, B., Kenny, T. (n.d.). *Open source software in the trenches: Lessons from a large-scale oss implementation.*

Fleiss, J., et al. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, *76*(5), 378–382.

Gelles, D., Waters, R. (2010, May). *Google ditches windows on security concerns.* Financial Times website. (http://www.ft.com/cms/s/2/d2f3f04e-6ccf-11df-91c8-00144feab49a.html)

Ghosh, R. (2005). *Free/libre/open source software in government.*

Ghosh, R. (2010). *Guideline on public procurement of open source software* (Tech. Rep.). iDABC.

Ghosh, R., Krieger, B., Glott, R., Robles, G. (2002). Open Source Software in the public sector: policy within the European Union. *Part 2b of the (F/L)OSS survey. Maastricht: Infonomics. Online: http://www. infonomics. nl/(F/L)OSS/report.*

Giera, J. (2004). The Costs And Risks Of Open Source. *Cambridge, MA: Forrester Research Inc.*

Gijtenbeek, M., Malipaard, G. (2004). *Uitgebreide geconsolideerde proof of concept - a7 project* (Tech. Rep.). OSOSS and Footmark.

Glynn, E., Fitzgerald, B., Exton, C. (2005). Commercial adoption of open source software: an empirical study. *IEEE transactions on software engineering*, *?*, 1-10.

Grand, S., Krogh, G. von, Leonard, D., Swap, W. (2004). Resource allocation beyond firm boundaries A multi-level model for Open Source innovation. *Long Range Planning*, *37*(6), 591–610.

Hahn, R. W. (2002). *Government policy toward open source software: an overview.* AEI-Brookings Joint Center for Regulatory Studies.

Hamel, G., Prahalad, C. (2005). Strategic intent. *Harvard Business Review*, *83*(7), 148.

Hauge, Ø., Sørensen, C., Conradi, R. (2008). Adoption of open source in the software industry. *IFIP International Federation for Information Processing*, *275*, 211-222.

Heur, B. van. (2010, April). *Heerenveen miskende complexiteit open office.* online webpage. (http://www.computable.nl/artikel/ict_topics/open_source/3332756/1277105/heerenveen-miskende-complexiteit-open-office.html)

Hippel, E. von. (2005). Open source software projects and "user innovation networks". In J. Feller, B. Fitzgerald, S. Hissam, K. Lakhani (Eds.), (p. 267-278). MIT press.

Hippel, E. von, Krogh, G. von. (2003). Open source software and the private-collective innovation model: Issues for organization science. *Organization Science*, *14*(2), 208-223.

Holck, J., Larsen, M., Pedersen, M. (2005). *Managerial and technical barriers to the adoption of open source software.*

Holst, W. van. (2010). *Modelteksten voor open voorkeur in een (europese) aanbesteding* (Tech. Rep.). NOiV. (https://wiki.noiv.nl/xwiki/bin/view/NOiV/Modelteksten+voor+open+voorkeur+in+een+aanbesteding)

Hoppenbrouwers, J. (2007). Community customers. In K. St.Amant B. Still (Eds.), (p. 510-522). Information science reference.

Hospital, B. (n.d.). Annual report 2008.

Hulsman, S. (2010, March). *St. antonius reduceert kosten met open source.* computable website. (http://www.computable.nl/artikel/ict_topics/open_source/3282801/1277105/st-antonius-reduceert-kosten-met-open-source.html)

Hwang, S. (2005). Adopting Open Source AND Open Standards IN THE Public Sector: Five Deciding Factors BEHIND THE Movement. *The Michigan Journal of Public Affairs*, *5*, 5.

Joo, Y., Kim, Y. (2004). Determinants of corporate adoption of e-marketplace: an innovation theory perspective. *Journal of Purchasing and Supply Management*, *10*(2), 89–101.

Jullien, N., France, B. (2009, July). Floss firms, users and communities: A viable match? *SSRN Working Paper.*

Kamphuis, A. (2009, November). *Wie is de baas?* Online, http://www.binnenlandsbestuur.nl. (http://www.binnenlandsbestuur.nl/vakgebieden/digitaal-bestuur/opinie/weblogs/wie-is-de-baas.147508.lynkx)

Kauffman, J., Li, X. (2003). *Payoff externalities, informational cascades and managerial incentives: A theoretical framework for IT adoption herding.*

Klomparends, L. (2006). *Open source software bij de publieke overheid* (Tech. Rep.). Vrije Universiteit Amsterdam.

Knubben, B. (2004). *Investeren in openheid. open source, wat kost dat nou?* online. (programma Open Standaarden en Open Source Software voor de Overheid)

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

Koh, E. (2009). *The adoption of open source software by singaporean companies.* Unpublished master's thesis, Queensland University of Technology.

Krishnamurthy, S. (2003). A managerial overview of open source software. *Business Horizons.*

Krishnamurthy, S. (2006). On the intrinsic and extrinsic motivation of free/libre/open source ((f/l)oss) developers. *Knowledge, Technology & Policy*, *18*(4), 17 - 39.

Krogh, G. von, Spaeth, S., Lakhani, K. R. (2003, July). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, *32*(7), 1217-1241.

Laffont, J., Martimort, D. (2002). *The theory of incentives: the principal-agent model.* Princeton Univ Pr.

Lakhani, K., Wolf, R. (2005). Making sense of the bazaar: Perspectives on open source and free software. In J. Feller, B. Fitzgerald, H. Scott, K. Lakhani (Eds.), (Vol. 1, chap. Why hackers do what they do). University of Maastricht: MERIT/Infonomics.

Landis, J., Koch, G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 159–174.

Larsen, M., Holck, J., Pedersen, M. (n.d.). The challenges of open source software in IT adoption: Enterprise architecture versus total cost of ownership.

Laszlo, G. (2007). Handbook of research on open source software: technological, economic, and social perspectives. In K. S. Amant B. Still (Eds.), (p. 445-459). Idea Group Inc (IGI).

Lee, J. (2006). Government policy toward open source software: The puzzles of neutrality and competition. *Knowledge, Technology & Policy*, *18*(4), 113–141.

Lee, Y., Kozar, K., Larsen, K. (n.d.). The technology acceptance model: Past, present, and future. *Communications of the Association for Information Systems (Volume 12, Article 50)*, *752*(780), 780.

Lerner, J., Tirole, J. (2002, June). Economic perspectives on open source. *Journal of Industrial Economics*, *50*(2), 197-234.

Lessig, L. (2002). Open source baselines: Compared to what? *Government Policy toward Open Source Software. Washington, DC: AEI-Brooking Joint Center for Regulatory Studies*, *1*, 69-86.

Markus, L. (2007). The governance of free/open source software projects: monolithic, multidimensional, or configurational. *Journal of Management and Governance*, *11*, 151-163.

Masrek, M., Jamaludin, A., Hashim, D. (2009). Determinants of Strategic Utilization of Information Systems: A Conceptual Framework. *Journal of Software*, *4*(6), 591.

Mathisen, R. O. (2008). *Adoption of open source software in the software industry.* Unpublished master's thesis, Norwegian University of Science and Technology.

Medicexchange. (n.d.). *Ireland entrenched to decide on pacs.* (http://www.medicexchange.com/PACS/ireland-entrenched-to-decide-on-pacs.html)

Microsoft. (2009, October). *The eula.* online. (http://www.microsoft.com/about/legal/useterms/)

Midgley, G. (2000). *Systemic intervention: Philosophy, methodology, and practice.* Springer Us.

Miles, M., Huberman, A. (1994). *Qualitative data analysis: An expanded sourcebook.* Sage Pubns.

Miller, J. (2002). Allchin's Folly: Exploding Some Myths About Open Source Software. *Cardozo Arts & Ent. LJ*, *20*, 491.

Miralles, F., Sieber, S., Valor, J. (2005). *CIO herds and user gangs in the adoption of open source software.*

Mockus, A., Fielding, R., Herbsleb, J. (2006). Two case studies of open source software development: Apache and mozilla. In J. Feller, B. Fitzgerald, S. Hissam, K. Lakhani (Eds.), (p. 163-211). The MIT press.

Modine, A. (2009, September). *Us bank dumps sharepoint.* www.theregister.co.uk.

Morgan, L., Finnegan, P. (2007). How perceptions of open source software influence adoption: an exploratory study. *Prodeedings of the 15th European Conference on Information Systems*, *15*, 973-984.

Munoz-Cornejo, G., Seaman, C., Koru, A. (2008). An empirical investigation into the adoption of open source software in hospitals. *International Journal of Healthcare Information Systems and Informatics*, *3*(3), 16-37.

Nagy, D., Yassin, A., Bhattacherjee, A. (2010). Organizational adoption of open source software: barriers and remedies. *Communications of the ACM*, *53*(3), 148–151.

Nichols, D., Twidale, M. (2003). The usability of open source software. *First Monday*, *8*(1), 21.

NOiV. (2007). *Actie plan nederland open in verbinding* (Tech. Rep.). Kabinet 2007. (https://noiv.nl/over-noiv/)

NOiV. (2010, April). *Nl octrooicentrum met open source voorloper bij de rijksoverheid.* www.noiv.nl. (https://noiv.nl/voorbeeldprojecten/nl-octrooicentrum-met-open-source-voorloper-bij-de-rijksoverheid/)

Oosterbaan, O. (2010, May). *âĂŸaanbesteding van oss binnen de overheid: wel fud, geen issue'.* online, accessed may 10th.

Ozel, B., Jovanovic, U., Oba, B., Leeuwen, M. van. (2007). Perceptions on floss adoption. *International Federation for information processing*, *234*, 319-324.

Pare, G., Wybo, M. D., Delannoy, C. (2009). Barriers to open source software adoption in quebec's health care organisations. *Journal of medical systems*, *33*, 1-7.

Pawson, R., Greenhalgh, T., Harvey, G., Walshe, K. (2004). Realist synthesis: an introduction. *ESRC Research Methods Programme Working Paper, University of Manchester.*

Pawson, R., Tilley, N. (1997). *Realistic evaluation.* Sage Publications Ltd.

Perens, B. (1999). The open source definition. In *Open sources: Voices from the open source revolution.* O'Reilly Associates.

Perens, B. (2005). *The emerging economic paradigm of open source* (Tech. Rep.). Consortium for Open Source in the Public Administration.

Plsek, P., Bibby, J., Whitby, E. (2007, March). Practical methods for extracting explicit design rules grounded in the experience of organizational managers. *The Journal of Applied Behavioral Science*, *43*(1), 153.

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, *12*(3), 23 -49.

Rogers, E. (1976). New product adoption and diffusion. *Journal of Consumer Research*, *2*(4), 290–301.

Rogers, E. (1995). *Diffusion of innovations.* Free Pr.

Romme, A., Endenburg, G. (2006, March-April). Construction principles and design rules in the case of circular design. *Organization science*, *17*(2), 287.

Rooij, J. de. (2010, February). *Amsterdam schuift open source op lange baan.* online webpage. (http://www.inoverheid.nl/artikel/nieuws/1959833/amsterdam-schuift-open-source-op-lange-baan.html)

Rosemann, M., Vessey, I., Weber, R., Wyssusek, B. (2005). *Reconsidering the notion of requirements engineering for enterprise system selection and implementation.*

Schiefl, F. (2010, March). *Quality over time in munich.* online blog. (http://www.floschi.info/2010/03/quality-over-time-in-munich/)

Seshadri, S. (2005). *Sourcing strategy* (S. Lorre, Ed.). Springer.

Sieverding, M. (2008). Choice in Government Software Procurement: A Winning Strategy. *JOURNAL OF PUBLIC PROCUREMENT*, *8*(1), 70.

Slabman. (2010, May). *Usage.* forum post on theregister.co.uk. (http://forums.theregister.co.uk/forum/1/2010/05/18/open_source_uk_government_what_next/)

Souabi, B. (2007). *Fabels & feiten over gesloten en open source software* (Tech. Rep.). Stichting ICTU.

Stallman, R. (1985, March). *The gnu manifesto.* (http://www.gnu.org/gnu/manifesto.html)

Stopford, J., Baden-Fuller, C. (1994). Creating corporate entrepreneurship. *Strategic Management Journal*, *15*(7), 521–536.

Szulanski, G. (1996). Exploring internal stickiness: Impediments to the transfer of best practice within the firm. *Strategic management journal*, *17*(1), 27–43.

Tornatzky, L., Klein, K. (1982). Innovation characteristics and innovation adoption-implementation: A meta-analysis of findings. *IEEE Transactions on engineering management*, *29*(1), 28–45.

Tranfield, D., Denyer, D., Smart, P. (2003). Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic. *British Journal of Management*, *14*, 207–222.

Tushman, M., Anderson, P. (1986). Technological discontinuities and organizational environments. *Administrative science quarterly*, *31*(3), 439–465.

Valimaki, M., Oksanen, V., Laine, J. (2005). *An empirical look at the problems of open source adoption in Finnish municipalities.*

Van Aken, J. (2005). Management research as a design science: Articulating the research products of mode 2 knowledge production in management. *British Journal of Management*, *16*(1), 19–36.

Varian, H., Shapiro, C. (2003). Linux adoption in the public sector: an economic analysis. *Working Paper*.

Ven, A. Van de. (2007). *Engaged scholarship: A guide for organizational and social research*. Oxford University Press, USA.

Ven, K., Huysmans, P., Verelst, J. (2007). The Adoption of Open Source Desktop Software in a Large Public Administration. *AMCIS 2007 Proceedings*, 501.

Ven, K., Van Nuffel, D., Verelst, J. (n.d.). The migration of public administrations towards open source desktop software: Recommendations from research and validation through a case study. *Emerging Free and Open Source Software Practices*, 191–214.

Venkatesh, V. (1999). Creation of favorable user perceptions: exploring the role of intrinsic motivation. *MIS quarterly*, *23*(2), 239–260.

Verheesen, M. (2010, January). *Review of literature on (free/libre) open source software.*

Waring, T., Maddocks, P. (2005). Open source software implementation in the uk public sector: Evidence from the field and implications for the future. *Information Management*, *25*, 411-428.

Webster, N. (1913). *Webster's revised unabridged dictionary* (L. Noah Porter D.D., Ed.). C. & G. Merriam Co.

Webwereld. (2008, Oktober). *Amsterdam zegt microsoft de wacht aan.* e-Business.

Weele, A. J. (2005). *Purchasing & supply chain management: analysis, strategy, planning and practice.* High Holborn House, 50-51 Bedford Row, London WC1R4LR: Thomson Learning.

West, J., Derick, J. (2006). Scope and timing of deplоment: Moderators of organizational adoption of the linux server platform. *International Journal of IT Standards Research*, *4*, 2.

Wheatley, M. (2004). The myths of open source. *CIO-FRAMINGHAM MA-*, *17*(10), 82–90.

Wheeler, B. (2004). Open source 2007: how did this happen? *Educause Review*, *39*, 12–27.

Wijnen-Meijer, M. (2007). *The acquisition of open source software* (Tech. Rep.). OSOSS (NOiV).

Wikipeadia. (n.d.). *Fleiss' kappa.* www.wikipedia.org. (http://en.wikipedia.org/wiki/Fleiss%27_kappa)

Wikipeadia. (2009, October). *Eula.* online. (http://en.wikipedia.org/wiki/Software_license_agreement)

Willems, W., Klip, D.-J. (2009, Juni). *De stand van zaken van het open source software beleid van de rijksoverheid.* Wilhemina van Pruisenweg 104, 2595 AN Den Haag, The Netherlands. (Deze implementatiestrategie is een uitgave van programmabureau NOiV)

Woods, D., Guliani, G. (2005). *Open source for the enterprise* (A. Oram, Ed.). O'Reilly Associates.

Xia, W., Lee, G. (2000). *The influence of persuasion, training and experience on user perceptions and acceptance of IT innovation.*

Yin, R. K. (1981). The case study crisis: Some answers. *Administrative Science Quarterly*, *26*, 58-65.

Yin, R. K. (1984). *Case study research, design and methods.* Sage Publications.

Zhao, L., Elbaum, S. (2003). Quality assurance under the open source development model. *Journal of Systems and Software*, *66*(1), 65–75.

Zuliani, P. (2004). Evaluating and Supporting OpenOffice.org. org in the Public Administration: the COSPA project.

Zuliani, P., Succi, G. (n.d.). An Experience of Transition to Open Source Software in Local Authorities. *Proceedings of e-Challenges on Software Engineering*.

Zwiekhorst, J. (2010, April). *Linux: kan office gemist worden?* www.computable.nl. (http://www.computable.nl/artikel/ict_topics/besturingssystemen/3311824/1277048/linux-kan-office-gemist-worden.html)