Free Software Patterns

Antonio Terceiro[1,2], **Rodrigo Souza**[1], **Christina Chavez**[1,3]

[1]Software Design and Evolution Group (aSide@UFBA)
Federal University of Bahia (UFBA)
Salvador, BA – Brazil

[2]Colivre - Cooperativa de Tecnologias Livres
Salvador, BA – Brazil

[3]Fraunhofer Project Center for Software and Systems Engineering
Federal University of Bahia (UFBA)
Salvador, BA – Brazil

{terceiro,rodrigo,flach}@dcc.ufba.br

Abstract. *The process of contributing to free software (aka open source software) projects has special characteristics that have fostered the emergence of many practices, each of which influenced by various forces and entailing trade-offs. The Free Software Patterns aim at documenting such practices by means of patterns, organized around three clusters: (a) Selection Patterns, that help prospective contributors to find suitable projects, (b) Involvement Patterns, that deal with the first steps towards getting familiar and involved with the selected project, and (c) Contribution Patterns, that document best practices for submitting different kinds of contribution to a free software project. The Free Software Patterns catalog is itself a free software project. Its license allows free reuse of the text, as long as the modified versions are distributed under the same license.*

1. Introduction

Free/Libre/Open Source Software (FLOSS) is a class of software projects that have Internet-based interaction between developers and public source code licensed under terms that comply with either the Free Software definition by the Free Software Foundation (FSF)[1] or the Open Source Definition by the Open Source Initiative (OSI)[2].

A FLOSS project starts when an individual developer, or an organization, decides to make a software product publicly available on the Internet so that it can be freely used, modified, and redistributed [Kon et al. 2011]. After an initial version of a FLOSS project is released and advertised in the appropriate communication channels, it may be subject to different kinds of contributions – for instance, development of new features, bug fixing, documentation – in which standard techniques and practices may be used, each of which influenced by various forces and possibly entailing many trade-offs.

Patterns are particularly well-suited for presenting and discussing techniques and practices. For instance, the Reengineering Patterns [Demeyer et al. 2008] present solutions for recurring software reengineering problems. Each pattern describes one part of

[1]http://www.gnu.org/philosophy/free-sw.html
[2]http://www.opensource.org/docs/definition.html

the reengineering process and produces different kinds of outputs, such as refactored code or insights into how the system functions [Demeyer et al. 2008].

Free Software Patterns present solutions for recurring problems that emerge when prospective contributors are willing to select an open source software project to get involved and to contribute with. These patterns entail activities that go beyond programming or reporting bugs. Therefore, *Free Software Patterns* also describe alternative ways of contributing to FLOSS projects, such as documenting, performing translations and writing tests.

Free Software Patterns are organized around three clusters of patterns. Figure 1 shows the clusters and their relationships. Each cluster is presented as a simple pattern language, conceived to document and to address a common set of problems that prospective contributors to FLOSS projects face:

- *Selection Patterns* help prospective contributors to find and select suitable FLOSS projects to contribute with.
- *Involvement Patterns* help prospective contributors to get familiar with FLOSS projects and figure out where to start.
- *Contribution Patterns* help prospective contributors to actually perform contributions to FLOSS projects. These patterns are organized according to the type of contribution, that is, documenting, translating, reporting bugs, resolving bugs, adding features, and so on.

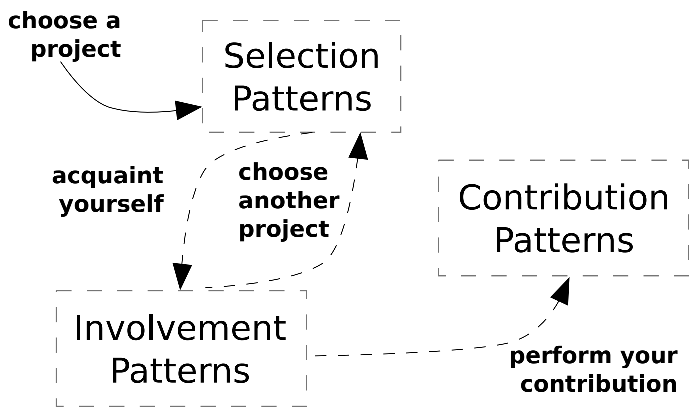

Figure 1. Free Software Patterns Overview

Selection Patterns and *Contribution Patterns* are intrinsically associated with the nature of FLOSS projects, and represent original work. *Involvement Patterns*, however, include two new patterns documented by us, but also reuse patterns from the Reengineering Patterns book, for instance, the *First Contact* patterns [Demeyer et al. 2008, p. 39], that help developers when they encounter a legacy system (not necessarily FLOSS) for the first time.

Selection Patterns (Section 2) and *Involvement Patterns* (Section 3) follow the approach to pattern description used in the Object-Oriented Reengineering Patterns (OORP) book [Demeyer et al. 2008]. The description of a cluster includes an overview, the forces that influence its set of patterns, and a map of these patterns, depicting how they may be related. For each pattern in the cluster, the following fields for pattern documentation are presented: pattern name, intent, motivation, problem, forces, solution, trade-offs, rationale, known uses, related patterns and next steps. The cluster description ends up with closing remarks and a brief discussion about its set of patterns.

Contribution Patterns (Section 4), however, are still work in progress and, therefore, they are presented as short descriptions of their proto-patterns, also known as *patlets*.

2. *Selection Patterns*

You have either been building systems with existing FLOSS projects or have been involved with using FLOSS, or have an interest for contributing to some existing FLOSS projects. You can have extrinsic motivations for contributing, such as improvement of programming skills, creation of required and unavailable code, and enhancement of professional status, or intrinsic motivations, such as altruism, fun, reciprocity, intellectual stimulations, and a sense of obligation to contribute [Oreg and Nov 2008, p. 2058].

When selecting a project, it is often the case that there are several repositories that list FLOSS projects, for instance, SourceForge[3] and Ohloh[4]. You navigate through some projects in these repositories but still have no clue about which FLOSS project could be a good starting point. You feel somewhat lost and concerned: there are several interesting projects, that use different technologies and programming languages, possibly with different development processes and types of participation. Is there a strategy for selecting a FLOSS project, that takes into account your motivations to contribute? How do you select a FLOSS project to contribute, aligned with your motivation?

2.1. Global Forces

The patterns in this cluster must resolve the following forces:

- *Basic Needs.* You need software to get your work done, and you do not have the resources to create them from scratch. Therefore, you have to find existing software that implements at least a large subset of the features you need.
- *Familiarity.* Previous use of a FLOSS project tends to facilitate your work and increase your motivation to contribute. You may be aware of existing features, bugs and enhancements that are necessary. This may ease your work as contributor.
- *Expertize on some technology.* You will possibly provide better contributions for projects that use languages and technologies that you happen to master.
- *Reputation issues.* Contributing to a FLOSS project implies in social interaction with other contributors in a new environment, and reputation-related issues and concerns may naturally arise. Reputation is gained by convincing your peers that you know what you are doing or talking about. This can be wonderful for you self-esteem and may also bring good jobs.
- *Feasibility.* It is self-rewarding to select a task, and spend some effort on it through its completion. You want to select a project in which performing contributions is feasible, that is, you will be able to successfully contribute and will be motivated to keep on contributing. Changeability, i.e. the ease of accommodating future changes, can be a major dimension on the feasibility of your contribution.

[3]`http://sourceforge.net/`
[4]`http://www.ohloh.net/`

- *Don't reinvent the wheel.* Think about a functionality. Probably, it has already been implemented by one or more open source software projects. Common wisdom recommends that you invest some time on searching for existing software that implements the desired features, before trying to develop a new software from scratch. Developing a software from scratch is not simple and requires time.

2.2. Overview

Selection Patterns help prospective contributors to find suitable FLOSS projects to contribute with, depending on some criteria. Figure 2 presents the three patterns that comprise this cluster.

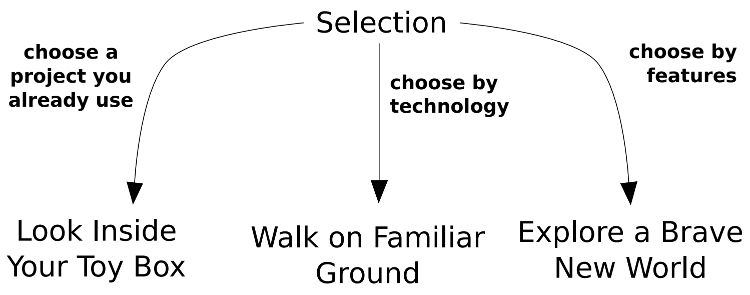

Figure 2. Selection Patterns

You should *Look Inside your Toolbox* (Section 2.3) to select an open source project that you already use and are somewhat familiar with. Or maybe you prefer to *Walk on Familiar Ground* (Section 2.4) and choose a project based on the technologies that stem as a familiar ground for you. Yet another possible path is to *Explore a Brave New World* (Section 2.5) whenever you are willing to choose a project based on the features or functionalities it provides, despite the risk of having to deal with unknown technologies.

2.3. Look Inside Your Toy Box

Intent: Choose a project that you already use to contribute with. For example, you got an innovative idea about some code editing feature not provided by any IDE and you are a frequent user of a particular IDE: why seek alternatives if you can contribute to the IDE you already use?

2.3.1. Motivation

You know that contributing to a free software project may help you develop or improve several relevant skills. These skills may be related to software development practice – programming, bug fixing, documenting, testing – or not – for instance, performing translations. However, you still have no clue about which project to contribute to, or do not have requirements for a specific feature or technology.

2.3.2. Problem

How can you start to contribute and then maintain your motivation high so that you can keep developing the desired skills?

2.3.3. Forces

- *Spinach can*. You want to develop stronger skills by contributing to a free software project.
- *Fear of the dark*. You may not feel comfortable contributing to a project that you know little about.
- *Feed two birds with one scone*. The desire to improve your skills may not be enough motivation for you to go on. Since you plan to invest your time contributing to a project, you may as well get additional benefits.

2.3.4. Solution

Choose a free software that you already use for contributing with, for example, your text editor, your web browser, your presentation software, your e-mail program, or your instant messaging program. If you also develop software, you may choose to contribute to one of the libraries or infrastructure software (compilers, IDE, etc.) you use. In any situation, make sure that the project you use is open source and accepts contributions from external developers.

Use your knowledge as a software user to *Write Documentation* (Section 4.2.1) or to *Translate To Your Language* (Section 4.2.2) important parts of the documentation. As a software user, you may stay tuned and *Write Useful Bug Reports* (Section 4.2.13) whenever you deal with errors.

However, to contribute by developing code or fixing bugs, you need to familiarize yourself also with the source code and with development issues. For instance:

- If you installed the software using a package manager, it's likely that you haven't even visited the project's website. Take your time to visit the the project's website, navigate throughout online material and *Skim the Documentation* (Section A.5) available to developers.
- If you installed the software using an installer, you probably don't have the source code. Grab the source code and *Do a Mock Installation* (Section A.2). You may also need to get the development files for the libraries that the project uses.

2.3.5. Trade-Offs

Pros:

- It might be easier to spot opportunities for contributing to a project that you already use. The contribution can be a piece of documentation that is missing, a translation, a feature that you need, or a fix for an annoying bug.
- Since many of your contributions will be useful to yourself (and to people with similar needs), you may feel more motivated to contribute.
- The more opportunities for contributing, the higher chance that you become successful and gain reputation.

Cons:

- By restricting yourself to projects that you already use, you may be missing the opportunity to contribute to projects that are friendlier to external contributors.
- You may overlook projects which are a better fit in terms of functionality.
- You may deal with projects that are complex and difficult to deal with.

2.3.6. Rationale

Developers "scratching their own itches" tend to be more motivated: the easiest, most straight-forward way to make a good contribution is to make something you want or need to use.

Of course, the nice thing is that once you contribute with anything, it is also available for others to use. If your contribution proves to be useful to other people, it is even better. After all, "given enough eyeballs, all bugs are shallow" [Raymond 1999], and you can use the community feedback to improve your contribution.

2.3.7. Known Uses

The second author was working on a system to automatically create and configure user accounts, in the beginning of each academic semester, for the students, professors and staff of a university [Cason et al. 2007]. However, sometimes it was necessary to create new user accounts in the middle of semester, and the original system was not user-friendly enough to be used on day-to-day operations by the support team. We were already using phpLDAPadmin[5], a user-friendly, web-based LDAP browser and manager, to browse and search user accounts. However, it did not provide a reliable mechanism to allow the creation of user accounts according to predefined configurations. Instead of creating a front-end for our system, we preferred to adapt phpLDAPadmin to our needs, using our knowledge as users, therefore contributing to the project.

Pinpoint[6] is an interesting presentation software: instead of requiring a WYSIWYG interface for designing presentations, it uses a simple plain text file format, what makes it perfect for keeping presentations under version control. After the first author started using Pinpoint for authoring slide presentations, some bugs led him to start contributing to the project.

Planner[7] is the GNOME software for project management. In a graduate course with an assignment that required selecting a free software project to contribute with, one team selected Planner because the members have been using it for several years. The students were concerned with Planner and its low-activity community. They thought that by contributing with Planner, software longevity could be extended.

2.3.8. Related Patterns

When you decide to Look Inside Your Toy Box, you will find one of two scenarios: either the project uses technologies you are familiar with, or it uses technologies that you would have to learn. In the former case, you are *Walking on Familiar Ground* (Section 2.4), so you should consider the pros and cons of such decision.

More likely, though, is that your toy box will lead you to *Explore a Brave New World* (Section 2.5). You should consider if you are willing to learn new technologies and deal with the problems that may arise when configuring your environment.

[5]http://phpldapadmin.sourceforge.net/
[6]http://live.gnome.org/Pinpoint
[7]http://live.gnome.org/Planner

2.3.9. What Next

As a user, you may have installed the software by using an installer or a binary package. Therefore, your environment might lack dependencies needed to develop the software (e.g., compilers and libraries). In this case, *Do a Mock Installation* (Section A.2) to make sure you have everything you need to get started with the code.

2.4. Walk On Familiar Ground

Intent: Choose a project based on the technologies it uses that stems as a familiar ground for you. For example, if you are familiar with a particular programming language, choose projects written in that language to contribute with.

2.4.1. Motivation

You are willing to contribute to a FLOSS project without having to learn or introduce unknown technologies in your working environment. It may be the case that you are very productive with certain technologies such as Java and MySQL. Or you are working with other developers, and most of them are using C# in existing projects.

2.4.2. Problem

How can you find a FLOSS project to contribute with based on technological constraints?

2.4.3. Forces

Choosing a FLOSS project to contribute with based on technology might happen for a variety of reasons:

- *You are looking for a project to be incorporated in an existing solution.* For example, you may be looking for a library to be used by an existing application. In this case, you need the library to be written in the same programming language as the application.
- *You do not wish or have the time to learn new things.* For example, your company has to add new functionality based on an existing project, but cannot afford the time required to get up to speed with new and unknown technologies and tools.
- *You may face organizational barriers.* Your organization has a strict policy for approving any new technology before it can be introduced, and you would rather avoid the trouble.
- *The adoption of a new technology might impose prohibitive costs.* For example, adopting an interpreted programming language for embedded software development will require adding extra storage capacity to devices in order to install the language interpreter.
- *You feel more confident.* Your programming abilities in a certain technology make you feel confident to overcome entry barriers and possibly perform better contributions to the project.

2.4.4. Solution

Look for projects that match your criteria for acceptable technology. Choose technologies that you are familiar with, that are approved by your organization or teammates, and that integrate nicely with your existing code.

If your criteria includes a specific programming language, you might start by looking for projects in language-specific repositories. For example, most Perl, Ruby and Python projects are listed in CPAN[8], Rubygems[9] and PyPI[10], their respective community repositories.

Sometimes your criteria will not be the programming language, but other parts of an application architecture, such as a database system. For example, you will want to check whether that new content management system you are evaluating supports the relational database system approved by your organization.

You can lookup projects in Freecode[11] (formerly Freshmeat), which is a catalog of Linux, Unix and cross-platform software, most of them open source. You can filter the list of projects by programming language and by tags, which are assigned by users. Some tags refer to particular technologies supported or used by the project, such as a database system or a particular software library. A similar service is Ohloh[12]. You can use it, for example, to look for content management systems written in PHP with support for MySQL databases[13].

2.4.5. Trade-Offs

Pros:

- The learning curve is minimized since you are dealing with known technologies or tools.
- Your environment is probably already configured for the project, or at least you know how to install the dependencies.
- You can make contributions of better technical quality since you are familiar with the technologies. Companies seeking programmers with particular skills can find potential hires by examining open source software code.
- You may be successful and gain reputation.

Cons:

- When you limit your choice by the technology criteria, you might exclude projects that would be a better fit in terms of functionality.
- If you always stick to familiar technologies, you may have trouble learning new tools that might prove themselves more useful in some contexts. For example, you could end up developing web applications in COBOL, although it lacks important libraries for web programming that are available for other languages.

2.4.6. Rationale

In general, many people are daunted by what they imagine is a high barrier to entry into a FLOSS world [Lester 2012]. Indeed, there is a barrier to overcome, possible as high as those ones you have to deal with when moving to a new school or getting a new job. In these situations, you certainly miss a familiar ground.

[8]http://www.cpan.org/
[9]http://rubygems.org/
[10]http://pypi.python.org/
[11]http://freecode.com/
[12]http://www.ohloh.net/
[13]http://www.ohloh.net/tags/cms/php/mysql

If you select a FLOSS project that uses technology you are familiar with, you can concentrate on other issues – for instance, understanding implemented features or getting comfortable with its development process.

Besides, you will probably provide better technical contributions to the project. When writing code, you may apply your knowledge about programming idioms, pitfalls to be avoided, supporting technologies and tools, guidelines that should be followed, etc. You may also find and fix bugs more easily, help on technical documentation, etc.

2.4.7. Known Uses

Back in 2006, we started working on a model transformation research project called TEMA. At that time, the majority of software technology for model transformation was in Java, but the developer involved in the project preferred Ruby. He looked for model transformation projects in Ruby, and found RMOF[14]. The project ended up contributing a substantial amount of code to the RMOF project.

One of the authors was looking for an extensible text editor to support his daily work. Also, he didn't want to spend too much time learning how to write extensions – the editor was supposed to support the work, not to prevent it from being done. Vim and Emacs, traditional text editors, were not good options, because Vim uses its own scripting language, and Emacs uses a variant of Lisp, with which the author is not familiar. The author ended up choosing an editor written in Python (with extensions also written in that language) and started contributing to existing extensions that provided most of the features he needed to support his workflow.

Chances are high that you and your teammates have already used this pattern many times. For example, it can be a web framework in PHP that you chose because you are familiar with PHP, or a library in Java that you chose because it integrates well with other libraries you are using. This is a very common pattern in practice.

2.4.8. Related Patterns

Sometimes, familiarity with the software is more important than familiarity with the programming language or other technologies. In this case, you might prefer to *Look Inside Your Toy Box* (Section 2.3).

If, on the other hand, functionality is more important than familiarity, you can always *Explore a Brave New World* (Section 2.5). However, you may discover that the project that best matches the desired functionality uses technology you are not familiar with.

2.4.9. What Next

After choosing a project using technology-based criteria, you should probably *Do a Mock Installation* (Section A.2) to reaffirm your certainty that the technology is under control.

2.5. Explore a Brave New World

Intent: Choose a project based on the features/functionalities it provides, despite the risk of having to deal with unknown technologies.

[14] http://rubyforge.org/projects/rmof/

2.5.1. Motivation

You are looking for a free software that provides a set of features or some functionality you need. Developing a software that implements such features from scratch may not be simple and requires time. Finding existing, running software that implements such features is therefore paramount, even if you have to learn new technologies it requires.

2.5.2. Problem

How do you find a free software project that meets you functional criteria and yet is suitable for contribution?

2.5.3. Forces

This problem is difficult because:

- *There may be several free software projects providing the functionality you are looking for.* These projects may involve different technologies and characteristics; it may be time-consuming to select a project that fits your requirements and that is also an easy target for your contributions.
- *Making a selection based on functionality only may not be trivial.* The functional aspects of any software project may be complex and hard to identify, and the need of having at least some familiarity with its use or technologies the software uses may not be neglected.
- *A free software project seems to implement the desired functionality but has some pitfalls.* For instance, there are several bugs to be resolved, changeability is a concern, and so on.

  Yet, solving this problem is feasible because:

- *You are eager to learn.* Learning is a common driving force for new undertakings. Contributing to a FLOSS project may bring several opportunities for learning new technologies, development processes, programming languages and even social skills. Learning new technologies can be a challenging and rewarding endeavor, despite the difficulties.

2.5.4. Solution

Focus on functionality rather than other aspects and look for free software projects that match your functional requirements. To focus on the functional requirements of the FLOSS project, you will need to:

- Prepare a list with those functional aspects that interest you.
- *Skim the Documentation* (Section A.5) of each candidate FLOSS project to identify the features it implements.
- Invest some effort on product assessment. When looking at each candidate project, answer these questions:
  - Does the FLOSS project satisfy your needs in terms of functionality?
  - If you want to integrate the software in a larger solution, how hard would that be? For example, if you are looking for a server software, does it have a client library available for the language in which your application is written?

  - Are the dependencies well documented?

When looking for projects that include some desired functionality, you can always do a web search. Then, by visiting the website for each project, you should be able to gather all information you need about functionality, dependencies, and integration. Sometimes the dependencies are not explicit in the website, but can be found in the source code package, inside a file such as `INSTALL` or `README`.

There are, however, other approaches to find relevant projects. Wikipedia[15] contains a lot of articles comparing software from particular domains along a set of attributes and features. The information is presented in tables (software vs. attribute), so it is easy to find software which include certain features. For example, the article "Comparison of web server software"[16] contains information about dozens of web servers, including their software license, operating system support, and support to features such as CGI, SSL, and IPv6.

Freecode[17], formerly Freshmeat, is a catalog of Linux, Unix and cross-platform software, most of them open source. You can use Freecode to find software by doing a textual search, filtering results by license, programming language, operating system, and tags. The page for each project contains a summary of the project, user comments, a list of recent releases, dependencies, links, and other information.

If you are looking for alternatives to a known project, you can use AlternativeTo[18]. You can restrict the results to open source software and apply filters for operating system and tag.

2.5.5. Trade-Offs

Pros:

- You have a higher chance of finding a project that fulfills your requirements in terms of functionality.
- You will get your job done in a lower amount of time with better chances to succeed.
- You will be probably learning a new language or technology.
- You may be successful and end up building your name in the community, that is, gain reputation.

Cons:

- You might end up spending a large amount of effort to get yourself up to speed with the technology used in the project.
- You may have trouble configuring your environment with all the dependencies (compiler, libraries, tools etc.), and you may not know where to ask for help.
- You may have a hard time to find a project, since the functional aspects of any software project may be complex and hard to identify.
- Other aspects, such as modifiability, if not taken into account, may turn you experience on contributing to a FLOSS project a nightmare.

[15]http://en.wikipedia.org/

[16]http://en.wikipedia.org/wiki/Comparison_of_web_server_software

[17]http://freecode.com/

[18]http://alternativeto.net/

2.5.6. Rationale

The need for using or adapting code that implements some functionality may be a highly motivating factor for selecting a FLOSS to contribute with. Possibly, the software will be ready for use with few adaptations.

Certainly, learning or taming a technology that you are not familiar with is a challenge, but it may also enhance your professional background and status. Empirical research suggests that self-development is highly important for software contributors [Hars and Ou 2002].

Finally, free software projects promote collaboration and other contributors can help you overcome your difficulties and enhance your technical knowledge.

2.5.7. Known Uses

Noosfero[19] is written in Ruby with some parts of the user interface written in Javascript, and everyone in the team was confortable with both Ruby and Javascript. When faced with the task of selecting a XMPP chat server for Noosfero, however, the team did not find an alternative in Ruby. Ejabberd[20] was a highly used XMPP server, but it was written in Erlang, a language that no one in the team had experience with. In terms of functionality, all the needed libraries for connecting to the server from both Ruby and Javascript were available. Ejabberd fulfilled all the needs in terms of functionality, but the team had to understand Erlang code in order to debug a couple of integration problems.

Analizo[21] is written in Perl and the authors found out that a source code parser was required. When looking into existing source code parsers, the authors have found that Doxygen[22], a documentation system for software projects, already supported parsing C, C++ and Java. Even though Doxygen was written in C++, it was used as a base for the source code parser used in Analizo, which was called Doxyparse. Creating Doxyparse required re-learning the venerable art of C++ programming.

Adium[23] is a free instant messaging application for Mac OS X that can connect to AIM, MSN, Jabber, Yahoo, and more. In a graduate course with an assignment that required selecting a free software project to contribute with, some students selected Adium because of its features. Adium is implemented in Objective-C and none of the students were familiar with that language, but they decide to challenge themselves and explore that new world.

2.5.8. Related Patterns

If to Explore a Brave New World proves itself a difficult journey, you can try the other patterns in this cluster. It might be easier, in terms of both effort and complexity, to *Walk on Familiar Ground* (Section 2.4). You may find projects that provide most of the desired features and still avoid dealing with unfamiliar technology.

[19]http://noosfero.org
[20]http://www.ejabberd.im/
[21]http://analizo.org/
[22]http://doxygen.org/
[23]http://www.adium.im/

Another approach is to *Look Inside Your Toy Box* (Section 2.3). You may end up having to learn new technology as well, but the learning curve is softer since you are already familiar with the project as an end user.

2.5.9. What Next

Since you do not necessarily know or use all the technology the new project requires, you might want start by *Doing a Mock Installation* (Section A.2) to make sure you have everything you need to get started with the code.

2.6. Discussion

It should be noted that the *Selection Patterns*, although presented separately for clarity, are not orthogonal. In fact, you can even use two of them together. For example, while *Looking Inside your Toy Box* (Section 2.3), you may select a project that uses unfamiliar technologies, and in this case you should consider the forces and trade-offs intrinsic to *Exploring a Brave New World* (Section 2.5).

Functionality is often an important criteria for selecting a project to contribute with, even if you intend to *Walk On Familiar Ground* (Section 2.4) or *Look Inside Your Toy Box* (Section 2.3). In the first case, functionality helps you choose among alternative projects that use the same underlying technology. In the second case, if the project you know as a user does not satisfy the required functionality, you may need to look outside the box.

Also, if a team is to select a project, the use of *Selection Patterns* may not be straightforward. Some of the team members may choose a project based on a familiar technology, although other members may still have to learn it. Likewise, the selected project may be used by some but not all team members.

3. *Involvement Patterns*

You have selected an open source project to contribute with, maybe one that you already use or that implements features you need. But it is a large project, with several thousands of lines of code, dozens of developers, bug tracking systems with lots of bugs to be fixed, and so on. You want to contribute with the project but first you need to get familiar with the software and get involved with the project ecosystem.

Are there any strategies to becoming familiar to a FLOSS project that should be considered? How can you get involved with a FLOSS project in a short amount of time, so that you can start to contribute as early as possible?

3.1. Global Forces

- *Software systems are often large and complex.* In such cases, it is not practical to try to understand the whole system before you start contributing.
- *Software evolves.* Even if you manage to fully understand the system, by the time you finish, the system would probably be different from that one you studied.
- *You are an outsider.* Chances are that nobody knows you in the project. Current developers may not trust you with big, important contributions until you show your abilities while contributing with small changes. Or maybe you need to build self-confidence before making contributions with a higher level of difficulty.

- *Contributions are welcome*. Usually, even small contributions are well received by free software projects, especially when they address issues recognized as important by the community.

3.2. Overview

Involvement Patterns deal with the first steps towards getting familiar with the selected FLOSS project and involved with its environment. Figure 3 presents the patterns that comprise this cluster.

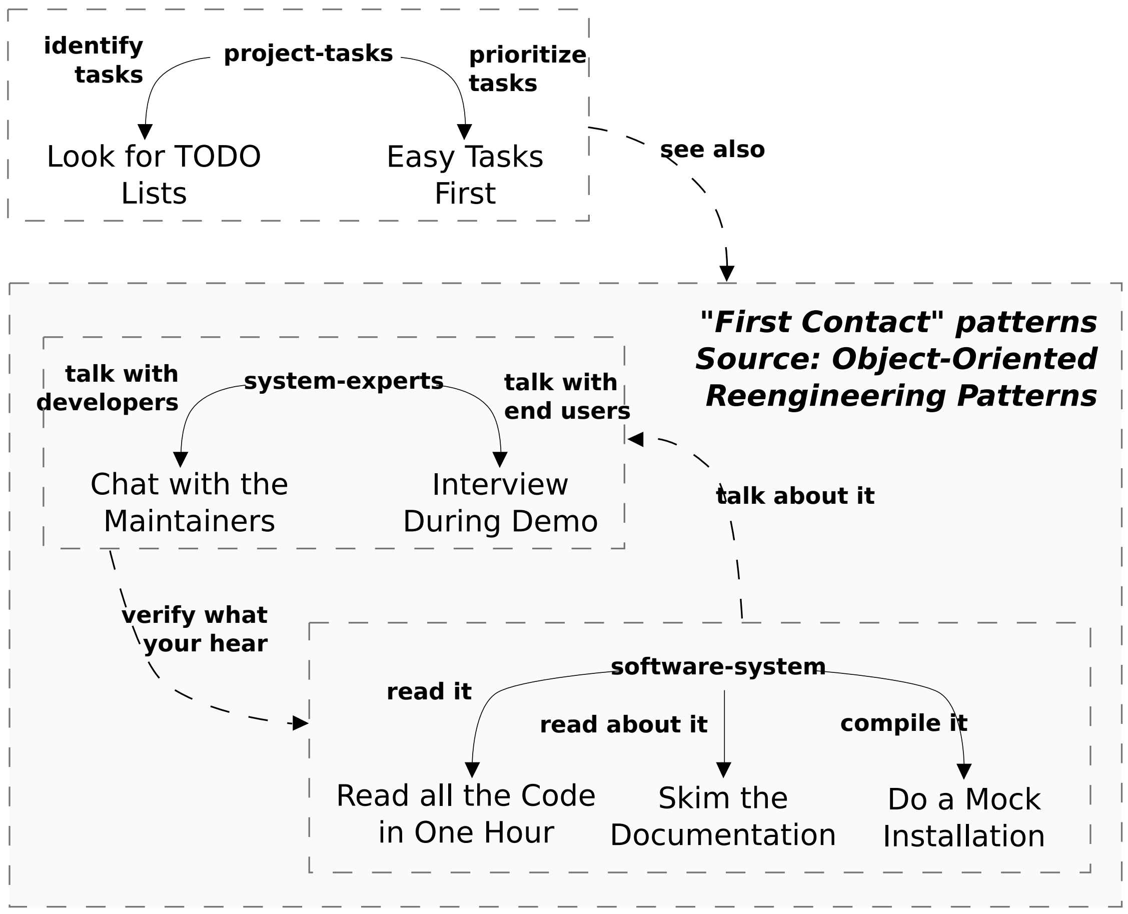

Figure 3. ***Involvement Patterns***

First Contact [Demeyer et al. 2008] consists of a set of patterns for reverse engineering that may be useful when you encounter a legacy system for the first time. FLOSS projects are legacy systems, and therefore these patterns may be useful for helping you to get familiar with a FLOSS system that is completely new for you. The intents of the patterns that comprise the *First Contact* cluster are listed in the Appendix A.

Look for Todo Lists (Section 3.3) and *Easy Tasks First* (Section 3.4) are patterns that can be useful when you are already getting familiar with the FLOSS project but still not sure about where and how you should start to contribute.

3.3. Look for TODO Lists

Intent: Discover places and things that may be good starting points for contributing to a FLOSS project.

3.3.1. Motivation

You are willing to contribute to some previously selected FLOSS project but have no idea of what to do or where to start. You used some of the *First Contact* [Demeyer et al. 2008] patterns and they helped you to get familiar with the software and some related artifacts. Now it's time to start contributing. However, you are still somewhat unfamiliar with the way things are done. For instance, if you choose a bug to fix or a improvement to make, it may be difficult to determine what needs to be changed in the code base.

3.3.2. Problem

How can you make a contribution if you are not familiar with the way things are done?

3.3.3. Forces

The first steps as a contributor can be hard for many reasons:

- *You have low experience with the project.* Because of that, you want to avoid working on high priority tasks.
- *You may not be able to complete tasks in a timely manner.* You have to provide your contributions in a small amount of time. If you delay your contribution, one of two problems can arise:
  - you may prevent other developers from doing their work, if their work requires that your task is completed; or
  - you may discover that another developer completed the task before you, due to its high priority.

  Yet, you make feel motivated to keep ahead because:

- *You may have instant gratification.* Possibly this will happen if you find trivial tasks to start with and manage to complete them in a reasonable amount of time.
- *Another brick in the wall.* Even responsible for minor or easy tasks, you feel you are part of something important.

3.3.4. Solution

Look for TODO lists in the FLOSS project. Many projects have TODO lists, i.e., lists of things that should be done eventually. These are usually low priority tasks, or tasks that no one has taken the time to think about seriously. TODO lists come in two flavors:

- Explicit TODO lists in a document (for example, a file named `TODO` in the project's root folder) that is bundled together with the application source code;
- Comments in the source code, marked by words or tags such as `TODO`, `FIXME`, `XXX`, or `???`.

Explicit TODO lists are usually plain text, in the form of itemized lists of tasks. Sometimes they are organized hierarchically, with items and subitems.

Most source code editors highlight occurrences of at least one of the markers, and some integrated development environments also create a list of such occurrences. You can also use the command-line utility `grep` to look for these markers in an arbitrary number of files.

The meaning for each specific marker is not well-defined. The following definitions[24] were provided by Martin Pool[25]:

> *There are useful distinctions between different tags. 'FIXME' is for things which are definitely broken, but where you want to not worry about it for the moment. 'TODO' is for useful features, optimizations or refactorings that might be worth doing in the future. XXX is for things that require more thought and that are arguably broken. Of course, you can make up your own categories, but I like these ones and they seem kind of standard.*

Martin also provided some examples:

```
/* TODO: How about auto-correcting small spelling errors? */
/* FIXME: This won't work if the file is missing. */
/* XXX: This method needs refactoring: should switch by core type. */
```

Source code documentation systems usually define special markers for annotating tasks in the source code. For example, Doxygen[26] creates a cross-referenced list with all occurrences of `\todo` or `@todo` inside source code comments. If the project uses a specific source code documentation system, check the system's documentation for a list of supported markers.

3.3.5. Trade-Offs

Pros:

- Source code comments usually refer to nearby code, so it is easier to understand what needs to be changed in order to resolve the issue.
- Source code comments usually refer to low priority, non critical tasks, so that you can manage to deliver your contribution on time.
- While looking for tasks in the source code, you may also be improving your understanding about the software after your *First Contact*.

Cons:

- Source code comments and TODO files, unlike bug reports, are not updated until the task is done. Therefore, they might contain outdated information (e.g., maybe someone is already working on a TODO).
- If the tasks are uninteresting you may lose interest in contributing.
- TODO lists may contain several tasks and you remain lost on how to start contributing.

[24]http://c2.com/cgi/wiki?FixmeComment
[25]http://c2.com/cgi/wiki?MartinPool
[26]http://doxygen.sf.net/

3.3.6. Rationale

Because TODO lists usually contain low priority tasks not assigned to any particular developer, chances are good that no one is working on the task, so you do not need to rush.

3.3.7. Known Uses

Many projects uses either explicit TODO lists, or special source comments, or both. JUnit[27] contains a file named `to-do.txt` with a itemized, hierarchical, list of tasks, but no TODO markers inside source code comments. On the other hand, Node.js[28] contains several `TODO`, `FIXME`, and `XXX` comments, but no TODO file.

CakePhP[29] is an open source web application framework written in PHP. It uses a tool for tracking issues that provides a list of tasks to be performed. This list is generated from source code annotations. The TODO's found in the source code are related to "methods that should be constructors, tests that should have been implemented, methods that could be refactored or even implemented".

The Code Style Guidelines for Android Contributors[30] recommends the use of TODO comments for "code that is temporary, a short-term solution, or good-enough but not perfect". When contributing to Financisto[31], an open source personal finance manager for Android platform, one of the authors used TODO comments to find code that needed enhancements.

3.3.8. Related Patterns

If you are not familiar with the FLOSS project, you should try the *First Contact* patterns [Demeyer et al. 2008]. Once you have identified some tasks from TODO lists, you should consider *Easy Tasks First* (Section 3.4). It may be a good idea to *Review Recent Activity* (Section 4.2.3) to make sure that your contribution is still relevant or that some other contributor did not perform it already.

3.3.9. What Next

After finding a suitable task for contributing with, you should consider taking a look at some of the *Contribution Patterns*. Be sure that you have the *Right Version for the Task* (Section 4.2.4), and *Review Your Changes* (Section 4.2.7) before committing them.

3.4. Easy Tasks First

Intent: Start working on easy tasks that can provide instant gratification.

3.4.1. Motivation

Start contributing to a FLOSS project can be daunting. You may be a novice programmer, that is used to develop small programs by yourself. Or, it may be the case that you have

[27]http://github.com/KentBeck/junit/
[28]http://github.com/joyent/node
[29]http://cakephp.org/
[30]http://source.android.com/source/code-style.html
[31]http://financisto.com/

some experience with software development but yet, real-world projects tend to be large and difficult to understand in its entirety, with possibly several tasks with different levels of difficulty and priority are waiting to be performed. You may even have no programming knowledge at all, but still want to contribute to a FLOSS project.

3.4.2. Problem

How can you start to contribute to a FLOSS project and yet keep interested?

3.4.3. Forces

Many forces are involved when you are preparing yourself to make your first contributions:

- *Analysis paralysis*. New contributors may be tempted to first understand everything and only then start to contribute. However, by the time a new contributor understands everything about a project – even if that could be possible – she might have already lost interest on it.
- *Shooting a moving target*. Software is continuously in process of change. If your contribution takes too long, it may be irrelevant by the time it is finished. For example, when you finish modifying a piece of code you realize that it has been replaced or removed in the latest version of the software.
- *Learning by doing*. Contributors can build their understanding of the system incrementally while they contribute.
- *Instant gratification*. Every contribution is important. When successfully accomplished, you feel like you're adding value to the FLOSS project, even if it is something trivial.

3.4.4. Solution

Look for easy tasks that need to be performed and start working on them. You will probably complete them and feel instant gratification. Easy tasks include working on bugs that have trivial fixes, adjusting the style of the source code to match the project coding style, or other housekeeping tasks.

Suggestions for easy tasks can be found in different places, but it is desirable that you try to find them by observing and interacting with the development ecosystem.

- First, you can *Look for TODO lists* (Section 3.3) and try to find easy tasks to work with.
- Blogs, wikis and websites may also provide clues on easy tasks for newcomers to start contributing.
- Mailing lists may be used to announce or discuss issues about the development of the project. Easy tasks can be identified there if other contributors that join these lists are willing to help, by providing suggestions for newcomers.
- Finally, IRC channels can be used to be in contact with other contributors and developers and easy tasks can be identified.

3.4.5. Trade-Offs

Pros:

- Working on easy tasks allows a contributor to gain knowledge about the system incrementally and at the same provide valuable contributions to the project.
- Dealing with easy tasks allows new contributors to focus on the process and not only on the tasks.
- Working on easy tasks provides instant gratification.

Cons:

- The contributor might feel that the easy tasks are not important enough because nobody did them before. However, it is useful to consider that dealing with the hard stuff may leave no room for thinking about the easy ones; also, sometimes small changes have great positive impacts on the project and on its users.
- Lists of easy tasks are not always actively maintained. If this is the case, you may find tasks that made sense in the past but are not applicable in the current state of the project. Be sure to signal your intent to solve a task to get updated information from developers.

3.4.6. Rationale

Besides knowing the code, there is also a lot of other aspects of a free software project that a developer can get used to by working on "trivial" tasks: getting used to the procedure for submitting contributions and having them reviewed by more experienced project members, interacting with the project infrastructure (bug tracking systems, communication channels etc) and others. By the time the new contributor has acquired the expertise to work on "more relevant" tasks (that are by definition also more difficult), the supporting infrastructure will not be an issue anymore.

3.4.7. Known uses

When the LibreOffice project[32] started, the initial team wanted to attract new contributors. They created an initiative called Easy Hacks[33], that provides guidance for new contributors and lists of tasks considered easy for newcomers to accomplish.

GNOME has a similar initiative called GNOME Love[34], which is "the place to learn how to start contributing to GNOME". Besides providing guidance on the topics contributors need to learn, GNOME Love also includes marking bugs that should be easy for new contributors to solve.

Noosfero has a list of tasks marked as Easy to Solve[35] where prospective developers can find bite-sized bugs to fix or enhancements to make.

[32]http://www.libreoffice.org/
[33]http://wiki.documentfoundation.org/Development/Easy_Hacks
[34]https://live.gnome.org/GnomeLove
[35]http://noosfero.org/Development/EasyToSolve

3.4.8. Related Patterns

You might want to *Look for TODO Lists* (Section 3.3) that already contain tasks categorized by difficulty level.

In general, you are dealing with easy tasks when you *Write Documentation* (Section 4.2.1) or *Translate To Your Language* (Section 4.2.2) important parts of the documentation.

3.4.9. What Next

After selecting easy tasks to start, you might want to look at some contribution patterns. Having the *Right Version for the Task* (Section 4.2.4) is crucial, and make sure you *Review Your Changes* (Section 4.2.7) before submitting them.

3.5. Discussion

Involvement Patterns describe strategies that can be used to walk the first steps in contributing to a project. The *First Contact* patterns [Demeyer et al. 2008] help prospective contributors in getting acquainted with the FLOSS product. The other two complementary patterns – *Look for Todo Lists* and *Easy Tasks First* – help them in getting partially familiar with the FLOSS process. However, the applicability of these patterns relies on specific resources that may or may not be found within a project, for instance, TODO annotations in source code and lists of easy tasks.

It is often a good idea to apply the *First Contact* patterns [Demeyer et al. 2008] to get familiar with FLOSS projects that are new for you or your team in a short amount of time. The patterns *Chat with the Maintainers* (A.1) and *Interview During Demo* (A.3) help you get acquainted with the people involved in conventional projects in the context of software reengineering [Demeyer et al. 2008]. However, while *Chat with the Maintainers* can be adapted for the FLOSS setting (for instance, by using IRC channels), *Interview During Demo* will not be of much help to get you acquainted with contributors and the FLOSS itself.

Finally, the use of *Involvement Patterns* can expose problems and risks related to the selected FLOSS project that may result in yet another round of using *Selection Patterns* (Section 2) to choose alternative projects for which contributions are feasible.

4. *Contribution Patterns*

Your desire to contribute to a FLOSS project is well underway. You selected a project, you became familiar with the legacy free software and you even identified possible starting points for contributing. How can you be sure that you will provide useful contributions, and that they will be accepted by project leaders? And what about your reputation if your contributions provide some undesirable effects to the project? Can you start with tasks other than coding or fixing bugs?

4.1. Forces

- *Fear of rejection.* You are concerned about possible barriers to entry into a FLOSS project or to have your contributions rejected by project leaders.

- *Lack of Confidence.* You lack confidence on your programming or other technical skills; this may be an hindrance on your ability to make "good" contributions.
- *Lack of Time.* Your available time may not be enough to allow you to make relevant contributions.

4.2. Overview

Contribution Patterns document best practices for submitting different kinds of contribution to a FLOSS project, such as, reporting bugs, discussing problems, writing documentation, performing translations, fixing bugs, creating tests, developing features, among others.

At the moment, the *Contribution Patterns* are work in progress. Therefore, in this section, we present them only as proto-patterns, with a name and intent. We are currently working on expanding and writing down these patterns in more detail and we plan to make them available in future publications. Figure 4 presents the patlets in this cluster.

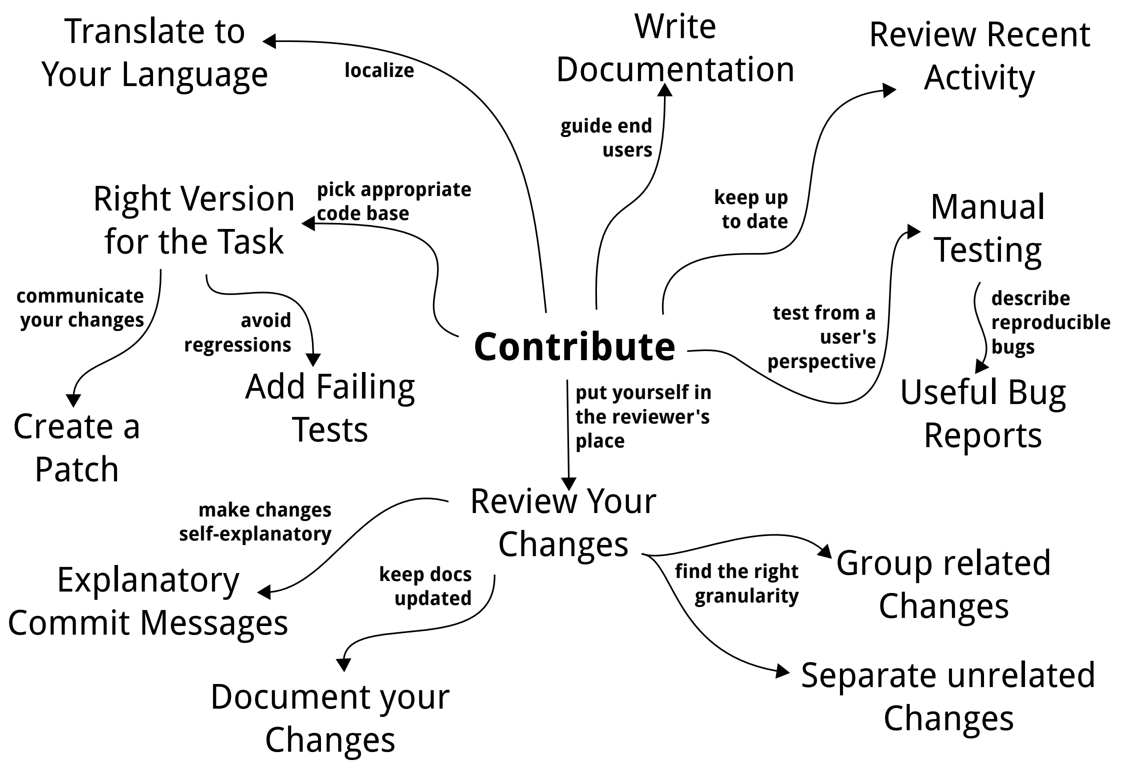
Translate to Your Language
localize
Write Documentation
guide end users
Review Recent Activity
keep up to date
Right Version for the Task
pick appropriate code base
Manual Testing
describe reproducible bugs
communicate your changes
avoid regressions
Contribute
test from a user's perspective
Useful Bug Reports
Create a Patch
Add Failing Tests
put yourself in the reviewer's place
Review Your Changes
make changes self-explanatory
Explanatory Commit Messages
keep docs updated
find the right granularity
Group related Changes
Separate unrelated Changes
Document your Changes

Figure 4. Contribution Patterns

4.2.1. Write Documentation

Contribute to a free software project by writing useful documentation.

4.2.2. Translate To Your Language

Contribute to a free software project by translating documents and software to your language.

4.2.3. Review Recent Activity

Identify news and opportunities related to the software project.

4.2.4. Right Version for the Task

Bug fixes in stable versions, new features in the development version.

4.2.5. Create a Patch

Produce a file that fully describes your changes to the source code of the project.

4.2.6. Add Failing Tests

Reproduce bugs in a automated way so that they don't come bac later.

4.2.7. Review Your Changes

Put yourself in the project leaders' shoes, and review your own work.

4.2.8. Explanatory Commit Messages

Make it easier for other people to understand the reasons for a change.

4.2.9. Document Your Changes

Update relevant documentation when adding new features or modifying existing ones.

4.2.10. Group Related Changes

Submit related changes as a single patch.

4.2.11. Separate Unrelated Changes

Submit unrelated changes in separate patches to ease the review.

4.2.12. Manual Testing

Test features by manually executing the software.

4.2.13. Write Useful Bug Reports

Provide all the information needed for other people to reproduce the bug you found.

5. Conclusions

In this paper, we presented the *Free Software Patterns*, which are organized around three clusters: *Selection Patterns*, *Involvement Patterns*, and *Contribution Patterns*.

These clusters are meant to guide potential contributors from project selection to involvement, and then to actual contributions. There is a strong interplay between Selection Patterns and Involvement Patterns: it is often necessary to get involved with a project in order to gather relevant information — e.g., quality of the documentation — before deciding if it is worth contributing to or if it is better to select another project.

Although free software projects tend to be open to participation, it is not always easy to find a suitable project and get involved in the community [Lester 2012]. This set of patterns should help potential contributors by documenting in a structured format some useful advices and best practices from free software communities.

We have briefly described an outline of the *Contribution Patterns*. We are currently working on expanding and writing down these patterns in more detail. As future work, we intend to refine the *Free Software Patterns* with the experience gained from applying them in an practical course on free software development.

The *Free Software Patterns* catalog is itself an open project. Its license allows free reuse of the text, as long as modified versions are made available under the same license. Anyone interested in sharing experiences and points of view is welcome to contribute with new patterns and refinements by visiting the official pattern repository[36].

A. *First Contact*, from Object-Oriented Reengineering Patterns [Demeyer et al. 2008]

A.1. Chat with the Maintainers

Learn about the historical and political context of your project through discussions with the people maintaining the system.

A.2. Do a Mock Installation

Check whether you have the necessary artefacts available by installing the system and recompiling the code.

A.3. Interview During Demo

Obtain an initial feeling for the appreciated functionality of a software system by seeing a demo and interviewing the person giving the demo.

A.4. Read the Code in One Hour

Assess the state of a software system by means of a brief, but intensive code review.

A.5. Skim the Documentation

Assess the relevance of the documentation by reading it in a limited amount of time.

Acknowledgements.

A very special “thank you” for Joe Yoder, for his invaluable availability, dedication and effort in shepherding us. We are also grateful to Serge Demeyer, Stéphane Ducasse and Oscar Nierstrasz for having licensed the Reengineering Patterns under the Creative Commons Attribution-ShareAlike 3.0 Unported license, which allowed us to partially base our own work on theirs. Last but not least, we would like to thank – for their supportive interaction and rich discussions during our lectures and meetings in 2012 – Mario Jorge Pereira, Rafael Glauber Silva, Sergio Gramacho, Luciana Silva, Debora Nascimento, Thiago Colares, Thiago Mendes, Vagner Amaral, Aline Meira, Patricia Melo, Murilo Botelho, Alvaro Lordelo and Tiago Motta.

[36]Source available at
`https://gitorious.org/flosspapers/free-software-patterns/`.

References

[Cason et al. 2007] Cason, D., Rocha, R., Terceiro, A., Barbosa, A., Ramos, E., and Galiza, H. (2007). Gerenciamento automático de usuários de uma rede acadêmica. In *Workshop Software Livre. Anais do Fórum Intern. Software Livre (FISL)*.

[Demeyer et al. 2008] Demeyer, S., Ducasse, S., and Nierstrasz, O. (2008). *Object-Oriented Reengineering Patterns*. Square Bracket Associates. (This book is available as a free download from http://www.iam.unibe.ch/ scg/OORP/.).

[Hars and Ou 2002] Hars, A. and Ou, S. (2002). Working for free? motivations for participating in open-source projects. *Int. J. Electron. Commerce*, 6(3):25–39.

[Kon et al. 2011] Kon, F., Meirelles, P., Terceiro, A., Chavez, C., Lago, N., and Mendonça, M. (2011). Free and Open Source Software Development and Research: Opportunities for Software Engineering. In *XXV Brazilian Symposium on Software Engineering (SBES 2011)*, Sao Paulo, SP.

[Lester 2012] Lester, A. (2012). 14 Ways to Contribute to Open Source Without Being a Programming Genius or a Rock Star. http://blog.smartbear.com/software-quality/bid/167051/14-Ways-to-Contribute-to-Open-Source-without-Being-a-Programming-Genius-or-a-Rock-Star.

[Oreg and Nov 2008] Oreg, S. and Nov, O. (2008). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Comput. Hum. Behav.*, 24(5):2055–2073.

[Raymond 1999] Raymond, E. S. (1999). *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition.