Implicit theories of “good leadership” in the open-source community

version 0.1.7

Master Thesis in Economy and Business Administration

By Gianluca Bosco

Department of Manufacturing, Engineering and Management
Technical University of Denmark
Supervised by Assistant Professor Kasper Edwards
Kgs. Lyngby
May 2004

Implicit theories of “good leadership” in the open-source community

Master Thesis in Economy and Business Administration
May 2004

This Master Thesis has been developed by Gianluca Bosco while hosted as a guest student by the Technical University of Denmark at the Department of Manufacturing, Engineering and Management.

Any comments or additional information regarding this paper are welcome. The raw data collected for this thesis is available on request to anyone interested in performing further analysis. If you think this paper is interesting or you plan to use it as a reference, I would be very happy to know about that.

email address: g.bosco@bitache.net
website: http://www.bitache.net/gbosco

Keywords: Free Software; Open Source Software; Leadership; Implicit Theories;

Acknowledgments

I would like to thank Kasper Edwards for inviting me at the Denmark Technical University and supporting me in pursuing my ideas concerning this thesis. Without his help and guidance, this work would not have been possible.

My appreciation goes to Poul-Henning Kamp of the FreeBSD project, and Kenneth Rohde Christiansen of the GNOME project, for kindly introducing me to the topic of leadership in the open-source environment.

I would like to thank all the open-source developers who accepted to participate in the questionnaires set up for this thesis, and for the interest they have shown.

A great “thank you” goes to my family, for fulfilling all my wishes.

At last but not least, I thank Anja, to whom I am mostly indebt. I will never forget the hours she spent keeping me company during the writing of this thesis.

Dedicated to my grandparents Lia and Bruno.

Table of Contents

Introduction

This thesis investigates the topic of leadership in the online communties of unpaid contributors who develop “open-source” software like the Linux kernel and the Apache web server. Such communities have lately been the focus of different studies. However, little research has been conducted on leadership in such environment. This thesis provides a contribution to this topic, focusing on the expectations contributors hold towards an open-source project leader. Specifically, the aim of this work is to identify the main factors describing the open-source contributors’ personal believes concerning the characteristics of a “good project leader”.

The paper is divided into two parts: 1) Existing knowledge, 2) Research objective and analysis.

Part 1 presents only existing knowledge. The first chapter contains background information on open-source software and the social-technical phenomenon behind its development. Understanding what open-source software is, why it is produced, and the customs of the community behind its development, is important to frame the discussion on leadership. Chapter 2 introduces the approach used to investigate the topic of “good leadership” in the open-source environment, and states why it is worth being studied. Chapter 3 reports an in depth description of the theoretical model chosen for this paper.

Part 2 contains all that is new, and is the result of the collection of empirical data and application of statistical analysis. Chapter 4 reports the early observations that inspired the aim of this thesis, the research objective and the steps undertaken to achieve it. Chapter 5 and 6 present both different stages of data collection and analysis. Chapter 7 and 8 present respectively a discussion of the analysis and conclusions.

Part I: Existing knowledge

1 Brief overview on open-source software

This chapter provides background information on open-source software and the social-technical phenomenon behind its development. Specifically, this chapter will present:

- What "open-source" software is.
- Who produces it, and why.
- What are the customs regulating the leadership dynamics in the open-source community.
- Some existing interpretations of the role of an "open-source project leader".

1.1 What is "open-source" software?

Computer software is what makes human beings able to interact with a computer. There are different types of software, providing different functionalities depending on the users' needs. Some software are used to type documents; others are used for entertainment purposes (e.g. videogames); still some others carry out even more critical functions, like managing bank transactions or controlling a little "Mars rover".

Software is created by writing a sequence of human-readable instructions, called "source code" (or simply "code"). In order to be used, the source code must be translated into a computer-readable sequence, the "binary". While the binary permits the usage of the software, access to the source code allows its modification.

The source code represents the value of a software. The source code of an application is like the recipe for the Coca-cola[1]. Anyone with the recipe of Coca-cola at hand would be able to mix it in the kitchen sink; anyone with access to the source code behind an application, would be able to appropriate the knowledge behind it. By no surprise, firms involved in the development of software treat the source code as a trade secret to protect.

Software usually comes with a license. The license defines the terms of use for the software, and states what the final user may do and may not do with the software in itself. A software license typically regulates the user's right to modify and redistribute it. Companies involved in the production of software make sure that their applications come with a specific type of license, to protect their investments. A traditional software license, usually addressed as "proprietary license", poses restrictions over the user's right to copy the software or to modify and redistribute it. Limiting users' ability to modify or redistribute the application is a way to limit appropriation of the

[1] I borrowed the "Coca-cola" analogy from the Ph.D. thesis of Kasper Edwards. See Edwards (2003).

investments returns. Hence, software covered by proprietary licenses is distributed only in its binary form, and no source code is provided to the user.

Software with this type of license is called “proprietary software”. The Microsoft “End-user license agreement” (EULA) is a proprietary license that accompanies e.g. “Microsoft Windows XP” hence a proprietary product.

Alternative to proprietary software is “open-source” software. This software is characterized by a different category of license called “open-source” license. The “Open-Source Initiative” (OSI)[2] defines the conditions[3] to be met by a license schema to be classified as open-source. It basically states that an open-source license must protect an unconditional right of any party to copy, modify and redistribute the software. In order to make it possible for the user to modify the software access to the source code must be provided. Hence, open-source software is distributed along with its source code.

The OSI maintains a list of licenses approved as “open-source”. Among them two types of licenses schemas deserve particular attention: the “General Public License” (GPL) and the BSD license. Both the GPL and BSD license provide access to the source code, but while the GPL requires derivative works to be redistributed under the same GPL license, the BSD does not. This means that while software with a BSD license might legally be turned proprietary, GPL’ed software might not.

It becomes clear that an open-source license provides weak protection to the creator’s intellectual property rights over the source code. If protection to intellectual property rights is weak, who would want to develop such software and why? We will see how the production of this software is typically carried out by online communities of volunteers often expecting no monetary reward at all.

[2] As the homepage reports at http://www.opensource.org : “Open-Source Initiative is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community specifically through the OSI Certified Open Source Software certification mark and program.”
[3] The conditions are listed into the “Open-Source Definition” (see http://www.opensource.org/docs/definition.php)

1.2 The size of the open-source phenomenon

It would be uninteresting to investigate how those open-source software are produced and why, if open-source was a small and isolated instance. Thus, it is worth to have a look at the size of this phenomenon.

SourceForge.net which provides free hosting / infrastructural services for open-source projects, maintains a counter of hosted projects and registered users. The main page of SourceForge.net by the 18 February 2004 reports 76.168 hosted projects and 793.653 registered users. Even though we cannot expect the SourceForge counter to be exhaustive of the real number of open-source projects around the Net, it still suggests that open-source might not be an isolated case, but the outcome of some conditions that tend to reproduce themselves on the Net.

"Open-source" becomes even more interesting if we think that almost the 70%[4] of the web servers around the Internet are running an open-source application named "Apache", available for free download (and modification) at http://www.apache.org. The Linux kernel in another sparkling open-source example, likely to be known also by those that are not very knowledgeable about computers. This software, which attracts not only the interests of regular users but also financial resources of big companies[5], has been developed through the efforts of more than 3,000 developers and a countless other contributors distributed over 90 countries on five continents[6]. Other famous examples are Sendmail which delivers the majority of the emails through Internet, and BIND, the standard solutions for Internet name resolution[7]. Other examples of open-source applications (that will be cited throughout this work) are the FreeBSD operating system, the PERL programming language and the GNOME desktop environment.

1.3 Who produces the software and how?

The most interesting aspect of the development of open-source software is that it relies on virtual online communities of unpaid volunteers. Those communities of volunteers gather around software projects organized on the Internet and contribute to their development by implementing new

[4] Netcraft web survey accessed the 18 Feb. 2004. http://news.netcraft.com/archives/web_server_survey.html
[5] As an example, IBM, HP, SUN, Compaq, SAP [...]. See the "Berlecon Research GmbH - FLOSS report - part 2: Firms' open-source activities, July 2002." Available at http://www.infonomics.nl/FLOSS/report
[6] Data taken from "The Essence of the Distributed Work: The case of the Linux Kernel" by Jae Yun Moon & Lee Sproull (2000); available at http://www.firstmonday.org/issues/issue5_11/moon/index.html
[7] BIND (Berkley Internet Name Daemon) is a DNS server: its purpose is to convert human-readable Internet addresses like into numerical ones understandable by a computer.

features, resolving problems, or simply testing the product. A large number of the participants contribute for free, without expecting any economic reward at all. A survey[8] conducted during 2002 reported that only the 15.7% of the respondents were directly paid to develop open-source software. The reader can imagine that there are as many contributors' communities as the number of existing open-source projects. Each community spins around a software project and its infrastructure - a project web page and mailing lists. The project web page contains either the links for newcomers interested solely in downloading and using the software and the necessary information for those interested in contributing actively to the development.

The development process of an open-source software can be described in the following manner: anyone can download the software along with the sources from the dedicated web page and freely use it. Since the source code is available, the same user might extend the software characteristics or simply fix bugs[9], provided he has the required programming skills. When the new characteristic (or fix) is developed, it is usually posted back to the community in the appropriate mailing list, to be integrated in the next software release. Once emailed to the mailing list, any subscriber can review the contribution and point out problems or enhancements if any!

It should be noted that there is a central figure called "project leader" or "maintainer" who is in charge for accepting, reviewing and integrating new features or fixes into the main source code. Once the project leader has reviewed and accepted the contribution, it is eventually integrated and delivered to the large public of users in the next software version.

Each user/developer can choose on which project area to contribute and how much effort to dedicate to it. The majority of the contributors work on the project a few hours a week, even though some others are much more involved. Almost the 70%[10] of the open-source contributors do not spend more than 10 hours a week in developing open code. Not all software users have the necessary skills to implement their own features or fix bugs; still, they can contribute by engaging in alternative activities, like translating documentation, testing the software and reporting bugs.

Project participation is fluid, since contributors are not bound by any employment relation. Anyone is welcome to join, choose on which area to work and what feature or fix to develop. Even though,

[8] See the "Berlecon Research GmbH - FLOSS report - part 4: Survey of Developers, June 2002." Available at http://www.infonomics.nl/FLOSS/report
[9] A software bug is an "error, flaw, mistake or fault in a computer program which prevents it from working correctly." (taken from http://www.wikipedia.com under the voice "computer bug")
[10] See note 3

open-source developers may contribute different projects simultaneously, the majority limit their activity into one or two projects at a time (29% and 27% respectively) [11].

1.4 Why individuals engage in open-source software development?

Production of software is a complex task, it requires knowledge of programming and a deep understanding of the problem, which the program is intended to solve. In addition, software development is time consuming, time that could have been used to generate an income. This aspect makes researchers wonder why people with programming skills would engage in open-source software production without seeking any protection for their intellectual contribution.

What emerges from the motivational studies carried out in the open-source environment is a complex system of forces. These motivational forces have been classified into intrinsic and extrinsic motivational drives.

Intrinsic motivation has been defined as the "doing of an activity for its inherent satisfactions rather than for some separable consequence. When intrinsically motivated, a person is moved to act for the fun or challenge entailed rather than because of external prods, pressures or rewards." (Ryan and Deci, 2000). The fun or the sense of creativity in engaging in a certain activity (in this case, coding) is at the heart of intrinsic motivation (Deci and Ryan, 1985). Central to Deci and Ryan's discussion of intrinsic motivation, is the inborn need of certain persons for feeling competent and self-determining in interacting in their environment.

Lindenberg (2001) points out another aspect of the intrinsic motivation. People can act on the basis of a principle. From the author's point of view, behavior may be triggered by the feeling that one must follow a particular rule, or norm of principle. Lindenberg argues that this kind of obligation-based behavior (i.e. to act appropriately) stems from a process of socialization. Hence, the level of social identification with a community can trigger a behavior consistent with the norms of the group. Lakhani and Wolf (2003) note that, in the open-source environment there is a strong sense of identification with the community; in the authors' words "the existence of canonical texts like "The New Hacker Dictionary" (Raymond, 1996), "The Cathedral and the Bazaar" (Raymond, 1999) and

[11] See the "Berlecon Research GmbH - FLOSS report - part 4: Survey of Developers, June 2002." Available at http://www.infonomics.nl/FLOSS/report

the "General Public License" (Stallman, 1999) have created a shared meaning of the individual and collective identities of hackers'[12] culture and responsibilities of membership within it".

On the other side, some contributors participate to open-source projects to indirectly satisfy their own needs. The core concept behind extrinsic motivation is to engage on certain activities to provide indirect satisfaction to the needs. Someone considering his job solely as a source of income necessary to buy food, pay the rent and enjoy summer holidays, is extrinsically motivated; "working" it's just a medium to satisfy one's needs. Consistently, Lerner and Tirole (2002) explain contributors' decision to participate in terms of a rational economical calculus - hackers contribute to open-source development as long as the net value between benefits and costs is positive.

Costs implied in developing open-source software are definable in terms of lost income deriving from freely revealing code and not employing time in an alternative and profitable manner. But what are the benefits arising from participating in open-source development? Some participants get paid for their contribution by companies whose business rely on open-source software. Other contributors are moved by career concerns: contributing open code is a good way to render visible their programming skills, and increase the opportunity to find a good job in the future (Holmström, 1999). Still others are moved by the desire to learn and improve their coding skills, from the community interaction.

The need for a feature (both in a work and non-work environment) is another important aspect that sustains participation. Studies conducted on the sources of innovation field show that some users have a strong incentive to personally develop solutions to their own needs and freely reveal their innovations (Von Hippel, 2002). In this case, the need for a particular software functionality and the existence of "sticky" information[13] govern the personal development of new software features; Von Hippel (2002) notes that contributors may choose to free reveal their innovations if benefits stemming from free revealing (i.e. those discussed above) are higher relative to those obtainable through protection of intellectual property.

[12] According to first definition encountered in the "Online Jargon File, version 4.4.7" accessed the 24 Feb. 2004, an hacker is "a person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary [..]". "The Jargon File" is available at http://www.catb.org/~esr/jargon/index.html

[13] Information regarding user needs and the context of usage, which is costly to transfer to an external manufacturer (Von Hippel, 1994)

The question remaining is: how much each motivational drive presented counts in explaining why hackers engage in the development of open-source software? Lakhani and Wolf (2003), administered a questionnaire to 637 open-source contributors involved in 287 different open-source projects, and asked them to select up to three statements that best described their motivation to participate to their “focal” open-source project.

Survey results give evidence to both intrinsic and extrinsic elements. Considering the extrinsic motivational drives, the 58% of the respondents stated to contribute to satisfy the need for certain software functionalities. The 41.8% deemed as important the opportunity to improve their programming skills.

Intrinsic motivation has considerable importance as well: 44.9% agreed with the statement ”Project code is stimulating to write”. For the 28.6% of the respondents, an important motivator was the sense of obligation to give something back to the community.

1.5 Project owners as project leaders in the open-source community

A specific type of software license scheme is classified as “open-source” if it matches the conditions contained in the “Open-Source Definition” cited in section 1.1. The core of the “Open-Source Definition” is that anyone must have the right to access and modify the software source code[14]. This means that anyone with enough interest and skills can duplicate the sources, modify them, and redistribute them over again under another software name. It has been noted by Raymond (1999) that in reality such “forking”[15] rarely exists. The open-source culture has “an elaborate but largely unadmitted set of ownership customs, which regulates who can modify the software, the circumstances under which it can be modified, and (especially) who has the right to redistribute modified versions back to the community” (Raymond, 1999).

Raymond (1999) notes that, these customs identify a sort of software “ownership” made by a set of exclusive rights and obligations over the project. This system of exclusive rights typically consists in deciding what new features to integrate in the next release of the software; redistribute the code to the community; empower maintainers of certain areas of code; step down and “pass the baton” to the next volunteer. Typically, the project owner is addressed as the “project leader”.

[14] See “Open-Source Definition” ver 1.9, points 2 and 3, at http://www.opensource.org/docs/definition.php
[15] “An open-source project “forks” when it splits up into two or more fractions, often trying to accomplish the same goals.” (Edwards, 2000).

Now the question remaining is how it is possible to acquire project ownership. The typical ways are to create a piece of software, acquire it or publicly take over the project when the owner disappears (Raymond, 1999).

Create a project is the first way to acquire its ownership. Regarding this aspect, Raymond recognizes interesting similarities with the Anglo-American common law theory of land tenure. In the Lockean theory of property, anyone can homestead a piece of never-owned land and acquire its property putting his work in it; in the same way, anyone can acquire ownership of a piece of software by creating it or making substantial contributions.

The “transfer of title” is an alternative way to acquire ownership over a piece of code; a project owner that has decided to step down for any reason, has the exclusive right (and obligation) to “pass the baton” to the next volunteer interested in acquiring its ownership.

A project can loose its owner, or just be abandoned. Again, anyone can take over the project improving and defending it. In this case hackers’ customs requires the potential new owner to contact the previous one and/or give public notice of his intentions on relevant places (newsgroups or mailing lists).

1.6 Maintainers as secondary project leaders

As already stated a project leader has the exclusive right to empower “maintainers of certain areas of code”. A maintainer is a contributor that has the responsibility over a sub-portion of the code (e.g. a module). As an example, the CREDITS[16] file of the Linux 2.4.0 kernel reports 375 maintainers of particular sub-parts of code!

A maintainer enjoys a certain level of autonomy over the code he maintains. He has the capacity to decide what new features or fixes to accept and integrate in the subsystem he has responsibility for; he has control over the implementation of the interfaces with the rest of the project (Raymond, 1999). Still, major decisions involving the project as a whole, are the responsibility of the main project leader.

[16] The “CREDITS” is a text file that typically comes with the source code, in the software package available for download. This file contains acknowledgments to those who participate to the project, along with a description of their contribution.

Contributors acquire maintainership of a project subsystem making substantial and continuing investment of time in it. The two typical paths to become a maintainer are to take responsibility for a certain part of the project or to take the role of the “lord high fixer”, characterizing and fixing many bugs (Raymond, 1999). As the same author notes, hackers like to say that “authority follows responsibility”. In this sense open-source projects work like meritocracies, in which “authority” stems from level of participation and competence: as Kenneth Rohde Christiansen, GNOME developer, puts it “the more you do, the more you have to say” (Interview with Kenneth Rohde Christiansen, 8 Oct. 2003).

The maintainer figure is pretty similar to the one of the project leader, but while the project leader has a wider span of authority to include the whole project maintainers’ competence is focused on a particular subsystem.

1.7 Project leader’s tasks

Lerner and Tirole (2002) provided an early discussion of a project leader’s tasks, observing three different successful open-source projects (Apache, Perl and Sendmail). Even though leadership is not the main focus of their paper, they have given a very interesting contribution to the subject, identifying four general leader activities that might account for the success of an open-source project.

The tasks identified by Lerner and Tirole are: to provide a vision; divide the project into smaller and well-defined tasks (modules); attract programmers and keep the project together.

According to Lerner and Tirole (2002), “a project leader provides a vision by assembling a critical mass of code to which the community can react. Enough work must be done to show that the project is doable and has merit”. They observe that the three cases they presented, all were posing challenging programming problems as premises that the projects were not at a “dead end”.

A project leader should not do all the work by himself, but leave challenging problems to the others - this is as much important as providing a vision (Lerner and Tirole, 2002). The intrinsic aspects of the motivational forces behind the open-source development (i.e. fun, sense for creativity) require the leader to not engage or control all the challenging coding activities.

The third task for a project leader is to divide the project into smaller and well-defined tasks, which individuals can tackle independently. Dividing the project into smaller modules lowers the need for coordination.

Finally, leadership must keep the project together by preventing it from forking. A project "forking" occurs when it splits up into two or different parts, often trying to accomplish the same goals (Edwards, 2000). Even though there are some natural forces that prevent a project from forking like the loss of economies of scale due to the creation of smaller communities, still some other factors might encourage this phenomenon. Lerner and Tirole (2002) note that contributors "may have conflicting interests as to the evolution of the technology" and that "ego concerns may prevent one faction to admit that another approach is more promising".

A key factor in lowering the probability of forking becomes trust in leadership. In the authors' words, "contributors must believe that there leader's objectives are sufficiently congruent with theirs and not polluted by ego-driven, commercial or political biases".

The Lerner and Tirole's contribution implicitly poses emphasis on the role of the leader as an explanation for the success of an open-source project. However, these observations have been criticized by Edwards (2000).

Edwards maintains that it is not the project leader who provides a vision, but the very same "assembling of critical code". In his words a "co-developer can examine and use the source code and become inspired to develop a new vision". Hence, it is not the leader the locus of the vision, but the very same contributor who can imagine all the evolutionary paths his fantasy allows!

Edwards' critic is subtle but very important. Recognizing that the vision does not stem from the leader, but from the software itself – push for an alternative understanding of open-source leadership; given that a project leader has little impact on the co-developers' vision – Edwards prefers to address the same role with a different label: project maintainer[17].

[17] The term "maintainer" as used in this context by Edwards, does not identify "responsibility over a sub-portion of the code" (see section 1.6). Edwards simply substitutes the term "project leader" with "project maintainer", to emphasize his alternative understanding of leadership with respect to Lerner and Tirole. However, the labels "project leader" and "project maintainer" are used somewhat interchangeably by hackers, to indicate project ownership.

At the same time, Edwards does not refuse to recognize the importance of a leader in an open-source project; on the contrary, he provides three different reasons that accounts for leader centrality (Edwards, 2001). The first reason is linked to the fact that 1) a software author retains property rights over the software *name* 2) the software name is like a “brand name” associated with a system of qualities and an existing user base. A leader accepting to integrate contributors’ new features under his software name, provides them with a way to seize on both the software image and the existing user base.

In addition, a project leader provides a coordination service to the participants of the project; he makes sure that contributors working on the development of the same functionality are in contact with each other and hopefully collaborate; he makes sure that the development goes smoothly and tries to ease up tensions between disagreeing developers.

Being a maintainer is also a matter of working out the necessary practicalities, among which: helping new users getting started in the use of the software, organizing software files in a logical structure, answering users’ questions and maintain the project homepage and mailing lists.

1.8 Shared leadership in the open-source community

The previous section implicitly stated that ownership rights over a software can be held only by a single person. Project leaders as single owners surrounded by a group of maintainers are typically addressed as “benevolent dictators” (Raymond, 1999). The label “benevolent dictator” comes from the fact that, a project leader is expected to consult with maintainers before going ahead with key decisions. Again, hackers’ customs can explain why a project leader is expected to rule through consensus: maintainers, with their work and dedication acquire ownership for the module they have responsibility for.

Actually, the “benevolent dictator” leadership arrangement does not depict the whole picture in the open-source environment, since some projects are owned by a group. In these cases, the exclusive rights of a project founder are exerted by a team through different organizational arrangements.

The Apache project is managed by a small group, which exerts the typical ownership rights previously described. This group is made up by co-developers each voting for each relevant

decisions concerning the direction of the project. The voting process is based on a minimal quorum consensus and is supported by a specific mailing list (Fielding, 1999).

It is interesting to see how the decision process is carried out in the Apache project. Each developer can vote on any issue concerning the project, by sending an email containing a "+1" (yes) or a "-1" (no). For code changes, three positive votes a no vetoes are required before the change is allowed to go into the main source. For other type of decisions, three "+1" and an overall positive majority is required. Anyone on the mailing list can vote, but only the votes casted by the Apache Group are considered binding (Fielding, 1999).

In the FreeBSD project, the "FreeBSD core team" is responsible for deciding the overall direction and goals. This "board of directors" promotes active contributors to committers[18], assigns people to well-defined "hats"[19] and is the final arbiter of decisions involving which way the project should be heading. Besides the core team there are other groups of individuals who are responsible for taking care of certain designated areas. Among those, the "Release Engineering Team" responsible for setting release deadlines and controlling the release process.

In the FreeBSD project, direct modification to the source code is granted to a large group of individuals called "committers". By the 1st Dec. 2002, 275 committers were listed in the CVS log (Saers, 2003). These are usually the most active developers who can integrate their own code or the code submitted by the developers who do not have this privilege (Saers, 2003).

All these examples identify leadership arrangements different from the one of a single project leader surrounded by a pool of maintainers. However, it is not rare to find the same pattern of single ownership previously described, even into those projects in which leadership is formally shared. As an example, in the FreeBSD project portions of the source code are managed by a maintainer. Saers (2003) describes the role of the maintainers in the FreeBSD project, in the following manner: "Maintainership means that that person is responsible for what is allowed to go into that area of the code and has the final say should disagreements over the code occur. This involves proactive work aimed at stimulating contributions and reactive work in reviewing commits" (Saers, 2003, par. 2.1.4 "Maintainership")

[18] A committer is a developer endowed with the privilege to directly modify the main source code, as hosted in an online repository.
[19] Synonymous of "role"

2 Approaching the concept of “good leadership” in the open-source community

This chapter outlines:

- The approach used to investigate the topic of “good leadership” in the open-source community.
- The reason why it is important to know what “open-source good leadership” is.

2.1 The many definitions and perspectives on leadership

As Yukl (2002) states, “the term leadership is a word taken from the common vocabulary and incorporated into the technical vocabulary of a scientific discipline, without being precisely redefined”. The lack of a precise redefinition has created a sort of ambiguity of the meaning of the term, typically used as a synonymous of authority, power, administration, control and supervision (Yukl, 2002). The academics involved in the organizational studies field define the phenomenon in different ways accordingly to the aspect that most interest them; as Stogdill (Bass, 1990) notes “There are many different definitions of leadership as there are persons who have attempted to define the concept”.

Still, it is possible to identify a common pattern between the proposed definitions of leadership; most of them maintain that leadership is a process in which one person exerts influence over other people to guide, structure, facilitate activities and relationships in a group or organization (Yukl, 2002).

It is very important to recognize the variety of definitions and perspectives on leadership since they reflect deep disagreement about the identification of leaders and the leadership processes.

2.2 Leadership is in the eyes of the beholder

It has been noted that although researchers may disagree on what leadership is, the general public seems to have less problems with this term. As Offermann, Kennedy and Wirtz (1994) put it, “individuals possess their own naïve, implicit theories of leadership, and are readily to determine their boundaries and characteristics”.

Implicit theories held by an individual are generally defined as “personal believes” guiding the understanding and interpretation of a certain phenomenon. Even though, implicit theories are not

supported by any scientific validation, they have been shown to govern subsequent behavioral expressions of an individual when interacting with his environment (Kelly, 1963).

Sternberg (1985) has investigated the implicit theories held by people, about the concepts of intelligence, creativity and wisdom. His results indicate that individuals have well-defined personal understanding of these three constructs and that *they use them in their ratings of both themselves and others.*

Actually, it is not unusual for these implicit theories to vary considerably from the definitions held by academics. Sternberg (1985) notes that the definition of “intelligence” held by the general public does not match the scientific notion of the term; having “goals” is deemed by the public to be characteristic of an intelligent person, whereas this attribute is not included by academic definitions.

Individuals have implicit theories of leadership as well. Research in this field has been carried out by Lord, Foti and Phillips (1982). Lord et al. (1984) have demonstrated that people have a personal conception of the characteristic of a leader, in terms of a set of personality traits and behaviors. According to their theory, leaders emerge in a group depending on their match to a prototype held by the followers.

The work of Lord et al (1982) focuses heavily on the perceptual process of the observer. Accordingly, this section is labeled “leadership is in the eye of the beholder” – to emphasize the importance of the very same followers in the emergence of leaders.

As we will see in the next sections, this is the approach I selected to investigate the topic of “good leadership” in the open-source environment; But why should we bother with the personal believes held by contributors, in understanding what leadership is in the open-source environment?

2.3 Followers of a leader or leader of the followers? An egg-chicken problem

Let us temporarily put a definition on the concept of leadership, for sake of clarity. Kutz and Kahn (1978) defined it as an “influence increment over and above the mechanical compliance with the routine directives of the organization”. Even though I will NOT use this definition throughout my work, this statement is useful to explain why I have chosen a contributor’s point of view to understand what “good leadership” is in the open-source environment.

If leadership consists of an “influence increment over and above a mechanical compliance”, what makes this influence possible? And whatever this “key-for-leadership” is, does it reside in the leader figure in itself, or in the followers?

Early studies intended leadership as something stemming strictly from the leader. The focus of the leadership process was the figure of the leader, who the leader was and what the leader did. The best examples summarizing this point of view are the personality and behavioral studies aimed to identify the characteristics of leaders[20]; these studies compared leaders and non-leaders on personality traits and behaviors but failed to establish any strong relationship between these variables and leadership. Among the many inadequacies of this type of approach, there is the lack of generalizability of the results across situations or even within the same situation (Gibb, 1954).

Hence, researchers started to recognize the importance of the very same followers and associated contexts, in explaining how leadership works. Hollander and Julian (1969) suggested that leadership was to be investigated also in the followers, and not only in the leaders. As Smith and Foti interpreted the contribution of Hollander and Julian – “individuals emerge as group leaders by fitting the shared conceptions of followers” (Smith and Foti, 1998).

Meindl (1993) criticizes approaches to leadership strictly connected to the figure of the leader, and calls for a “reinvention of leadership from a radical social-psychological point of view”. In his contribution, he noticed that current leadership studies, approach leadership in terms of a process that the leader ultimately controls. Even though “mutual influence processes between leaders and followers are recognized, leadership is mostly conceptualized as something [...] to be performed by the leader. [..] It is dispensed to, and used on, followers to influence and control them” (Meindl, 1993).

Meindl, alternatively considers leadership as an experience followers undergo. Without this experience (as the authors says: “without being in state of leadership”) followership does not exists. Therefore, he proposes a more follower-centered approach to understand why and how leaders emerge.

[20] See Bass (1990) chapter 4 – “Traits of leadership: 1904 – 47”

2.4 Do leaders exist into those communities of volunteers?

There is a question we should pose to ourselves before wondering what “good leadership” is in the open-source environment: do leaders exist into those communities involved in the production of free software? Is their role recognized and accepted, by the community?

There are good reasons to question the existence of leaders into those communities of volunteers, or at least to question the willingness of an open-source participant to accept the presence of leaders. As we have seen in the first chapter, the sense of self-determination and autonomy plays an important role as an intrinsic motivator. Poul-Henning Kamp underlines that when he was part of the FreeBSD core team back in 1992-2000, much of the discussion focused on whether there should have been leadership for the project (interview with Poul-Henning Kamp, 10 Oct. 2003); as a reason for these discussions, he speculates that FreeBSD was seen as a sort of “boss-free heaven where people could relax”, away from their work environment; as he summarizes the thoughts of the participants “we are just all minding our own little bits, and nobody can tell us what to do” (email from Poul-Henning Kamp, 22 Feb. 2004).

Kasper Edwards notes that the term “leader” is used in an improper manner, when addressing an open-source project; in his article Edwards states that “the term "leader" seems a poor choice, since in the open source community most people are working for free. No leader in an open source project has the equivalent power of a leader in a private or public company. These real leaders possess the power to discharge people or in other ways affect the employee’s situation.” (Edwards, 2000).

Again, the problem resides in the meaning we attach to the term “leadership” and use to investigate such phenomenon.

My assumption for the current work is that leaders in the open-source community exist, to the extent to which the label “leader” is used (or at least, understood) by a generic contributor. The “leader” I am referring to is the project owner/project leader described by Raymond (1999). The real problem is NOT to question the existence of leaders in the open-source environment, but to uncover the shared meaning attached to it, by the community.

Hence, the leader I am addressing occupies an informal organizational position, in virtue of the rights granted by the customs of the community. These rights are typically summarized by the

leader’s exclusive ability to redistribute modified version of a software back to the community (see section 1.5).

2.5 Why is it important to know what a “good project leader” is?

It is reasonable to speculate that a style of leadership consistent with the expectations of those volunteers contributing free code can have an effect on the productivity of a project. The need to keep contributors satisfied with their project leaders is even more salient in an environment in which participation is not regulated by any employment relationship.

Although hackers’ customs place a sort of “taboo” against forking, such phenomenon sometimes occurs. Sometimes It happens that a “faction” of the community decides to follow an alternative evolutionary path for the software; in such cases the original project splits up in two or more branches, each one under different name. As the software splits in different and independent fractions, also the community efforts do. A style of leadership not consistent with the expectations of the community can account as a reason for a project fork; as an anonymous contributor commented in participating in the online questionnaire set up for this work:

“The project [the anonymous contributor participates to] is led in a top-down, traditional (closed-source) management style which does not fit free software projects. In the terms of the over hyped 'The Cathedral and the Bazaar'[21] document, the project is a “Cathedral” but claims to be a “Bazaar”. The words are there, but the actions are missing or misdirected. This has led to several forks of the project, many disillusioned contributors and a lack of progress. I will fork the project as well. The main reason for the fork is the unwillingness of the 'core developers' to accept the validity of outside contributions. “(Anonymous).

As the same contributor notes, another consequence of an inconsistent style of leadership is the lowering of the level of participation to the project;

“[bad leadership] ... has led to a total stop of external contributions, and also a more or less total stop of progress in the project. Free software thrives in an open environment. A closed-minded, hierarchical "honor-your-superiors"' environment work stifling and leads to fragmentation.” (Anonymous)

[21] “The Cathedral and the Bazaar” is a description-from-the field of the principles that drive a successful open-source project, written by Eric Raymond (1999).

As another participant pointed out, “bad leadership” can account for “hurt, anger, confusion” and hence the well-being of the community. Poul-Henning Kamp, reported in an interview situations in which certain types of leadership can account for much of the unproductive discussions in the mailing lists – and notes how those events could affect the spirit of the entire group (Interview with Poul-Henning Kamp, 10 Oct. 2003).

Therefore, knowing the behavioral expectations held by contributors towards leaders has a practical value: understanding these expectations is a first step towards analysis and training towards better management practices, in the open-source environment.

2.6 A theory for understanding perceptions of “good leadership”: traits and behaviors

The current work will use a theoretical model developed by Lord, Foti and Phillips (1982) and revised by Lord, Brown, Harvey and Hall (2001) on the implicit theories of leadership held by individuals. Lord et. al (1982) used a cognitive approach to uncover how leaders are selected and evaluated by individuals. According to their model, leadership is the outcome of a classification process unconsciously carried out by a generic observer; a leader emerges over a group to the extent to which he is categorized as such by the followers.

Leadership resides in the minds of the followers as a set of personality traits and attributes, called prototype. Prototypes are abstractions of the human mind used to classify persons, and develop consistent behavioral expectations towards them. Whenever a match between the perceived personality of a person and a prototype occurs, a consistent process of classification follows. When classified into a certain category, the behavior of the same person is interpreted accordingly and a process of development of behavioral expectations occurs.

As an example, we all have an idea of what the personality traits of an extravert person are. Typically, an “extravert” is described as someone very talkative, smiling, and outgoing. Whenever we meet a new person showing these traits, it is likely that we unconsciously classify him as an “extravert”. Accordingly, we develop some consistent behavioral expectations towards him: an extravert is likely to be invited to the next party on Saturday, to make the evening more fun!

To better understand this process of classification and the subsequent development of behavioral expectations, let us think about the feeling of surprise likely to arise whenever we discover that a “2 meters-tall guy, weighting 120 Kg” has a high-pitched voice. This is because we learned from experience that the characteristics “2-meters-height"/"120-kg-weight” are more likely to be associated with a deep voice, instead of a “high-pitched” one. If we think more about this process, we may discover that observers infer traits and characteristics from clues, without really experiencing them.

Accordingly, Lord et al. (1984) demonstrated that a prototype of a leader does exist in the mind of the persons. A leader prototype is seen as dedicated, goal-oriented, informed, charismatic, decisive, responsible, determined, intelligent and believable [...][22]. Subsequent observations (Lord et al. 1984), showed that individuals matching more closely this prototype, were more likely to emerge as group leaders.

Building on the model developed by Lord et al. (1982), this work will identify the main factors representing the characteristics and behavior of the prototype of a “good project leader” hold by open-source contributors.

[22] The leader prototype identified by Lord et al. (1984) consists of 59 characteristics. Lord et al. (1984) demonstrated that not all the characteristics have the same weight in the process of identification of leaders, carried out by a generic observer. The sample of attributes presented here, is made up by only the first 9 characteristics that have been shown to be most important in the selection of leaders.

3 Cognitive categorization and leadership perceptions

This chapter provides an overview on the cognitive dynamics governing the perceptions of leadership. Specifically, this chapter will explain:

- What are person categories, and what is their task.
- How person categorization occurs.
- The nature of the person prototypes used in the categorization process.
- The theory of leadership perceptions developed by Lord, Foti and Phillips (1982).
- The revision of the cited theory developed by Lord, Brown, Harvey and Hall (2001) which better accounts for the contextual constraints affecting leadership perceptions.

3.1 Overcoming memory limitations through knowledge structures

The human mind can be described as an information processing system, which elaborates stimuli coming from the external environment. Information about surrounding objects, persons or events are inputs for such system; conversely, human behavior is the output of such processing.

This system is generally thought to consist of several components like memory and attentional resources. Memory and attentional resources are scarce resources since they are not sufficient to elaborate accurately each stimulus coming simultaneously from the environment. As an example, unexperienced employees find it very difficult to type a letter and concurrently follow a discussion.

In order to overcome such processing limitations the human brain relies on “knowledge structures”. The term “knowledge structures” is used to refer to cognitive schemas otherwise known as scripts, categories, implicit theories, prototypes. Those structures are held in memory and allow people to *select*, *interpret*, *simplify* and *integrate* environmental information. The utilization of such knowledge structure lowers the need for attentional resources and permit and higher amount of information to be processed at the same time.

3.2 Person categorization

As Rosch, Mervis, Gray, Johnson, and Boyes-Braem (1976) note “the world consists in a infinite number of discriminably different stimuli”, placing high attentional demands for an observer. Consequently, the same authors recognize that “one of the most basic functions of all organisms is the cutting up of the environment into classifications by which nonidentical stimuli can be treated as equivalent”. This process of classification, called by cognitive psychologists “categorization process”, consists in grouping external stimuli into a category. By category is meant a group of objects considered equivalent, which are identified by a label (Rosch, 1978). As an example, the

label “dining room chair” identifies a category of objects which share certain type of attributes e.g. four legs, back, seat, [...].

As it has been noted by Cantor and Mischel (1979) humans tend to categorize not only common objects but also people based on personality attributes, physical appearance, gender, race, social occupation and general behaviors.

Categorization allows one to simplify and reduce a potentially overwhelming number of stimuli, thus lowering the need for attentional resources; this process occurs selectively focusing attention on certain aspects of a particular object or person, e.g. shape and functional attributes for objects, personality traits and behaviors for persons. Once categorized under a common label, the observer can predict specific features of any of the category members on the basis of general expectations about the category.

As an example to understand what a persons category is and how the categorization process works, let us think about a newsgroup “newbie”[23]. Every day someone is classified as “newbie” in a newsgroup by other readers, in a public or private manner, in a conscious or unconscious way. This classification is primed by a set of behaviors deemed by the observer as typical of a “newbie”. If we observe what primes categorization into the category “newbie”, we may notice that such classification occurs usually when someone:

- asks obvious questions in a newsgroup;
- uses an improper terminology to address a technical problem;
- uses a reverent tone in the firsts public postings;
- states himself to be a newbie;
- ..
- [place behaviors as *you* think they fit]

Once the person is classified as “newbie” subsequent behavior is interpreted accordingly to the information for that category. As an example, an experienced reader answering to a newbie asking for help in a technical newsgroup, is more likely to suggest very obvious alternatives on how to solve the problem, since newbies are expected to be unexperienced and quickly get stuck by the

[23] As the on-line version of the hacker Jargon File 4.4.7 states, a “newbie” is an “Usenet neophyte”, and thus indicates a new joiner of a newsgroup.

smallest inconvenient. As an alternative example, someone perceived as an “experienced hacker” is expected to provide good clues on how to resolve a complex coding problem.

These examples are meant to give a general idea on how the process of person categorization occurs. Indeed, it must be noted that each reader can have his own implicit theory of how a newbie behaves. These implicit theories are likely to vary considerably from the one I presented. Moreover, to make things even more complicated, each behavior listed can account for different types of person classification: someone “using an inappropriate terminology to address a technical problem” can be classified as a “wannabee”[24] instead, dependently on the context and on other behaviors shown.

However, the aspects above will be addressed in the next sections. What matters now is how categorization lowers the need for attentional resources, which occurs in two ways: 1) cutting down the number of attributes necessary to be classified into a category 2) allowing the perceiver to access systems of relevant information, learned through experience, to interpret and develop behavioral expectations.

3.3 Categorization occurs through prototypes

In the precedent section, we have seen how categorization process lowers the need for attentional resources. Basically, the observation of few meaningful behavioral “hints” coming from a person, primes the inference of a system of relevant information. This system of information, retrieved from memory, allows an observer to interpret accordingly the behavior of a person and develop a consistent set of expectations.

Cognitive psychologists address these sets of “meaningful primes” as “prototypes”. A person prototype is an abstraction of the human mind, which consists in a set of personality traits, personal attributes and behaviors. Prototypes play an important role in the categorization process since they are the “yardstick” by which a person is placed in a certain category; whenever a match between an external stimulus and a prototype occurs, an accordingly categorization follows.

The question now is, “what makes a specific behavior or attribute “meaningful” in the process of categorization?”

[24] According to the on-line version of the hacker Jargon File ver 4.4.7 a “wannbee” is a “would-be hacker”.

3.4 Principles governing the creation of prototypes

In order to understand how prototypes are created, we need to know about two basic principles that govern the categorization process. These principles, which explain what influences the content of a prototype, are called the principle of the "World Perceived Structure" and the principle of the "Cognitive Economy" (Rosch, 1978).

The first principle of categorization maintains that the world is not a set of equiprobable co-occuring attributes. Indeed, there are certain *environmental regularities* around which categories (and prototypes) are created. As Rosch exemplifies it "it is a fact that wings co-occur with feathers more than with fur". At the same time, we can say that, it is a fact that "using an improper terminology, when asking technical advise in a newsgroup" co-occurs more often with "being not skilled at resolving an easy technical problem".

The second principle of categorization states that the human mind attentional resources are limited. This means that the human mind tries to categorize objects or persons, with *minimum effort.* In the quest for "minimum effort for categorization" not all behaviors have the same importance. Some behaviors are more distinctive for a category and thus have a higher potential to distinguish around stimuli. Therefore, human memory will retain those primes which are more distinctive for a category as components of a prototype[25].

The more an attribute is shared by members of a category, and is rare among the non-members of the same category, the more it is prototypical. As an example, "owning a Ferrari" is a personal attribute which is particularly distinctive to categorize a person as "wealthy"; on the other hand, "having a credit card" is not really a good clue to classify someone in the same category.

The fact that attributes of a prototype can be qualified as "more or less" prototypical, raises the following question: what are the boundaries of a prototype?

3.5 The fuzzy nature of prototypes

Categorization occurs whenever the perceived qualities of a person match an existing prototype. This simplified description of how categorization works may implicitly transmit the idea that,

[25] Hence, the term "distinctive" is used as a synonymous of "prototypical"

classification is the outcome of a checklist of *necessary and sufficient features*. Consistently, a person must be perceived to have all the necessary characteristics to be classified into a category; if the person does not fulfill the required attributes – he will not be categorized accordingly. This assertion views categories as well-defined and distinct.

Actually, this “all-or-none” point of view does not correctly represent the nature of a person category. As Cantor and Mischel (1979) noted “.. [for an observer] it would be difficult to find a set of necessary and sufficient features shared by all members of any particular person category that one would want to use as the definitive test of category membership”. As an example, they note how classification process depends heavily on contextual constraints. Consequently, they suggest that the nature of person categories is best described by the concept of “fuzzy sets” whose borders are ill-defined. The existence of linguistic hedges like “almost”, “virtually” or “sort of” (Lakoff, 1972) is compatible with this conception of ill-defined categories. As an example, a robin and a penguin are both technically birds. Nevertheless, we may agree that while a “robin is a bird par excellence” a “penguin is a sort of bird”.

As Wittgenstein (1953) (cited by Rosch, 1978) has pointed out, categorical judgments are a problem only if one is concerned with boundaries – in the normal course of life, two neighbors know on whose properties they are standing, without exact demarcation. Thus, Rosch (1978) proposed an alternative conception of prototypes – as “the clearest cases of category membership defined operationally by people’s judgments of goodness of membership in the category”.

3.6 Effects of attributes prototypicality on items recall order

Prototypicality affects efficiency measures of human mind processing performance. In particular, two dependent variables are presented: speed of information processing and order and probability of item output (Rosch, 1978)

Speed of information processing - reaction time. The speed at which subjects can judge whether a stimulus (object or person) is part of a category, is a widely used measure which correlates with prototypicality. More prototypical items are associated with lower response time (Cohen, 1983; Lord et al., 1984).

Order and probability of item output. Rosch (1975) demonstrated that subjects are more likely to recall as first the most prototypical items, while less prototypical ones are cited in the last positions or not cited at all.

3.7 The theory of leadership perception (1982)

Lord, Foti and Phillips (1982) have built a theory of leadership perception on the work carried out by Rosch (1978) on object categorization and by Cantor and Mischel (1979) on person categorization. Their theory of leadership perception maintains that leaders emerge over a group to the extent to which followers perceive them as such. As Lord and Maher summarize the core of the theory:

> “The locus of leadership is not solely in a leader or solely in followers. Instead, it involves behaviors, traits, characteristics, and outcomes produced by the leaders *as these elements are interpreted by followers”* Lord and Maher (1991, pag. 11)

According to their theory, leadership is the result of a perceptual process. Leaders emerge on their match to the leader prototype held by the observers. Whenever a match between observed behaviors and a leader prototype occurs, an accordingly process of categorization follows. Once categorized, leaders are able to perform those functions consistent with the expectations held by the followers.

Empirical tests for the theory of leadership perception have shown:

- The personality traits and personal attributes associated with the prototype of a leader (Lord, Foti and De Vader, 1984; Offerman, Kennedy, Wirtz, 1994).
- The effects of leadership cognitive categorization in *selecting, evaluating* and *describing* leaders (in order: Cronshaw and Lord, 1987; Bartol and Butterfield, 1976; Eden and Leviatan, 1975)

The leader prototype discovered by Lord et al., (1984) is composed by 59 attributes characterized by different levels of prototypicality. The prototypicality was measured by asking respondents to rate each item out of a list of personality attributes depending on their fit to their image of a leader. Table 1, is a sample of the first 34 personality traits, sorted by prototypicality:

1	Dedicated	**18**	Fair
2	Goal oriented	**19**	Strong character
3	Informed	**20**	Open-minded
4	Charismatic	**21**	Persuasive
5	Decisive	**22**	Interested
6	Responsible	**23**	Insightful
7	Intelligent	**24**	Understanding
8	Determined	**25**	Competitive
9	Organized	**26**	Cooperative
10	Verbal skills	**27**	Loyal
11	Believable	**28**	Educated
12	Directing	**29**	Industrious
13	Good administrator	**30**	Caring
14	Honest	**31**	Humanitarian
15	Concerned	**32**	Persistent
16	Disciplined	**33**	Likeable
17	Trustworthy	**34**	Well groomed

Table 1 - The personality traits and attributes of the leader prototype discovered by Lord, Foti and DeVader (1984), sorted in decreasing order of prototypicality.

Laboratory tests showed how leader prototypes influence the process of *selection* of leaders (Cronshaw and Lord, 1987; Smith and Foti, 1998). In the experiment conducted by Cronshaw and Lord, groups of undergraduate students were shown a 12 minutes videotape, showing a target person interacting with other members of a work group. The videotape shown was different for each group participating to the experiment. In particular, in each videotape the target person behaved to match different sets of prototypical (or anti-prototypical) qualities. Leadership perceptions, measured through the "General Leadership Impression"[26] questionnaire administered to the students, were consistently higher relatively for those videotapes showing high prototypical behaviors.

Categorization affects *evaluations* of leaders, as well. As an example, there is overwhelming evidence that leader's gender affects their evaluations. In an experiment conducted by Bartol and Butterfield (1976), students were provided with one of two versions of an evaluative questionnaire containing four stories depicting different leadership styles. These leadership styles differed in the leader's orientation towards production emphasis, organizing behaviors, person consideration, and tolerance of freedom; Managers names were altered in the two versions, to indicate males or females. Results showed that leader gender has an effect on leader evaluations. Students scored female managers higher on person consideration, while male managers were scored higher in organizing behaviors.

[26] The GLI (General Leadership Impression) asks subjects to rate on a 5-point scale a) the amount of leadership the ratee exhibited b) how willing the rater would be to choose the ratee as formal leader c) how typical the ratee was of a leader c) to what extent the ratee engaged in leader behavior e) the degree to which the ratee fir their image of a leader. These measure are composed to produce a composite GLI indicator (Cronshaw and Lord, 1987, pag. 100).

The implicit theories of leadership substantially bias the way followers *describe* their leaders (Eden and Leviatan, 1975; Rush, Thomas and Lord, 1977). Rush, Thomas and Lord showed that there is a high level of congruence between factor structures obtained from descriptions of a fictitious supervisor and descriptions of real leaders. Authors concluded that since practically identical factors structures emerged from fictitious and real leaders descriptions, the actual behavior of a leader is relatively unimportant for behavioral descriptions, since descriptions are based mainly on prototypes. In particular, this biasing effect of prototypes has questioned the validity of some leadership measuring techniques relying on rating questionnaires administered to subordinates.

3.8 Relation between personality traits and observed behaviors

So far, the terms “personality traits” and “behaviors” have been used somewhat in an interchangeable manner. In order to understand how categorization process of persons works, a distinction should be made between these two concepts.

Indeed, we can imagine an inferential relation between behaviors and personality traits perceived by an observer; behaviors are observable “hints” that individuals use to infer personality traits and thus to classify people. As Calder puts it, “Judgments about leadership are made on the basis of observed behaviors. An individual who labels another as a leader has no direct knowledge of the other’s internal qualities. The other person’s behavior must serve as evidence for the existence of these qualities.” (Calder, 1977). Figure 1, shows this concept:

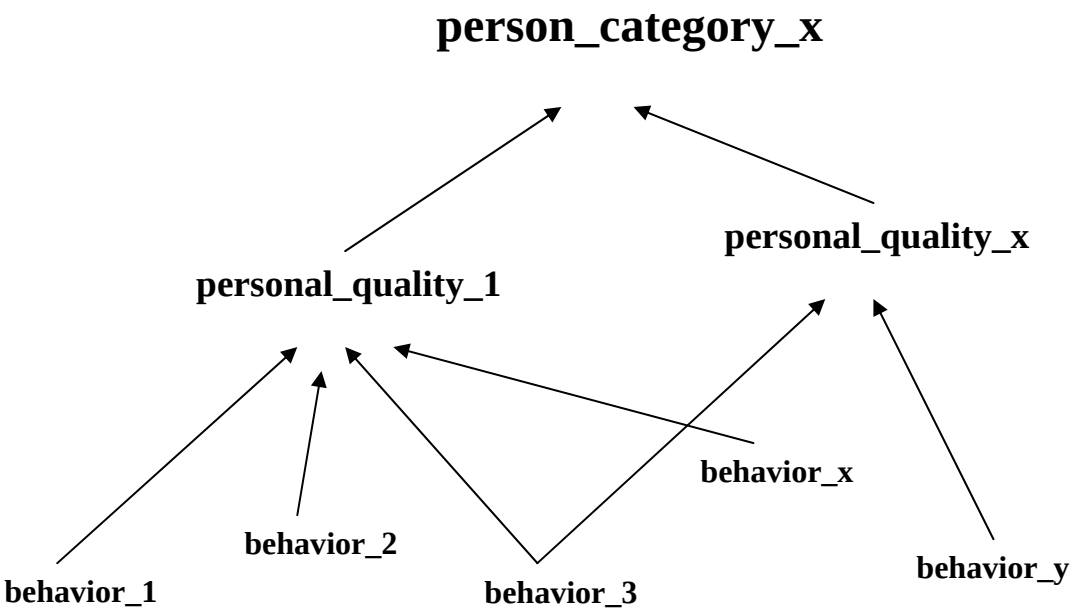


Figure 1 - Inferential relation between observed behaviors and personal qualities.

As an example, how hackers classify "good coders"? A reader knowledgeable about programming may realize that – indeed, there are some implicit theories used to evaluate the skills of a coder, in a conscious or unconscious way. As it emerges from some public postings available on www.google.com[27], "good code" is code understandable with minimum effort, easily modifiable, without bugs, and able to fulfill its tasks efficiently. In addition, the same behavior can account for different types of personal qualities: using "UPPER CASES" in a public posting will be understood as "shouting" and thus typically as a sign of impoliteness; on the other side, upper cases may be used to infer the authoritarian and inflexible tone of an email.

This simplified overview of the relation between personal traits and behaviors does not represent how the human mind technically process information. What is important to note is that personal traits are *summary labels* for certain sets of behaviors. These summary labels are easily communicable among individuals, and therefore are easily accessible to discover person prototypes.

3.9 Inferential processes and leadership perceptions

So far, leadership perceptions have been explained as a process of recognition of "appropriate" leader's behaviors and personality traits, as they are summarized by a prototype held by the observer. This process of traits and behaviors recognition can occur in a conscious or unconscious

[27] Information using the keyword "what is good code?".

manner. However, leadership perceptions have been shown to be influenced by *assessments of causality for organizational outcomes or events*, as well (Lord and Maher, 1991). In a study conducted by Rush, Phillips, and Lord (1981), people were asked to make leadership ratings, after having seen a 15 minutes video of group-solving problem. After viewing the video, but before making leadership ratings, bogus performance feedback was provided. Results showed that, positive performance feedback indeed reinforced leadership perceptions. Inferential processes of leadership perceptions can occur in a conscious or unconscious manner, as well. Observers may assess carefully leader causality for positive outcomes or events, evaluating the impact of facilitative or inhibitive environmental factors; this type of inferential process is compatible with the concept the observer as a “rational information processor”. Conversely, if we treat individuals as “limited information processor”, it is reasonable to maintain that they assess causality in a more automatic and unconscious way, through superficial and heuristic evaluations.

3.10 Reconstructive models of leadership prototypes (2001)

The traditional model of leadership categorization (Lord et al., 1982), conceived prototypes as *fixed* systems of information to be retrieved from the memory when needed. A prototype was considered a “file in the drawer” to be located whenever a good match with observed behaviors occurred.

Indeed, there is evidence suggesting that a fixed conception of prototypes might not represent correctly their nature. The main reason is that prototypes have been shown to vary extensively within the same individual dependently on many factors, such as context (e.g. military leaders vs. religious leaders) (Lord et al., 1984), hierarchical level (e.g. upper vs. lower level management) (Lord et al., 1984), national culture (Gerstner and Day, 1994), and task type and target gender (Hall, Workman and Marchioro, 1998). Hence, if prototypes were fixed “files in the drawer”, the human mind would contain an improbable high number of them to provide sufficient flexibility in perceiving leadership (Lord, Brown, Harvey and Hall, 2001).

As Lord et al., (2001) note, the context sensitivity and the extensive within-individual variability of leadership prototypes is more consistent with a model in which prototypes are generated on-the-fly, to satisfy contextual constraints (Barsalou, 1983).

Hence, it has been proposed that prototypes are not fixed systems of information to be retrieved from, but abstractions *to be rebuilt at occurrence* to correspond the requirements of different contexts, tasks or maturational stages of a group or organization (Lord et al., 2001).

3.11 Contextual constraints on prototype generation

The meaning of the term "leadership" is extremely sensitive to the context in which the perceiver experiences it. To understand this concept, let us think about the meaning of the term "LOVE"[28]. We all know what "LOVE" means; at the same time, there would be little doubt in admitting that the term "LOVE" has different meanings dependently on the context (e.g. "LOVE" for a sport, "LOVE" for your partner, "LOVE" for your nation [...]). Accordingly, Lord et al. (2001) maintain that the prototype used for evaluation and selection of leaders is affected by different contextual forces, among which:

- National and organizational culture
- Task nature
- Leader qualities
- Followers affective tone
- Followers self-schemas
- Followers level of identification with a group

Even though those constraints are simultaneously active, we can expect them to be characterized by different levels of relative strength in shaping the generation of the leader prototype; while in some situations the national culture may play a strong role determining the appropriate leader prototype, in others – follower's image of the self can have more weight.

3.12 External constraints: culture, task, leader qualities

Open-source software is produced by volunteers contributing from all over the world. Almost the 90% of the open-source developers that participated to a survey[29] funded by the EU-Commission, represented 21 different nationalities, from all the continents. The multi-cultural aspect of the open-

[28] I borrowed the "LOVE" example from "Contextual constraints on prototype generation and their multilevel consequences for leadership perceptions" (Lord, Brown, Harvey, and Hall, 2001).
[29] See the "Berlecon Research GmbH - FLOSS report - part 4: Survey of Developers, June 2002." available at http://www.infonomics.nl/FLOSS/report

source development, make us wonder whether it could account for differences in leadership perceptions. Gerstner and Day (1994), showed how business leader prototype varied considerably among groups of participants of different nationalities, and underlined the potential effect on practices of intercultural management.

The national cultural layer is not the only cultural "force" likely to have an effect on prototype generation. As Lakhani and Wolf (2003) note, the open-source community shares its own culture, represented by canonical texts like "The Jargon File", "The Cathedral and The Bazaar" (Raymond, 1999) and the "General Public License" (Stallman, 1999); the existence of different and overlapping cultural layers poses another question, related to the relative strength of each cultural layer in respect to leadership.

The same nature of the task and leader qualities may have an impact on prototype generation, as well. As an example, Hall et al. (1998) showed how, in small groups, females were more likely to emerge as leaders on consideration tasks (e.g. behaviors aimed at the well-being of the group) while males were more likely to emerge as leaders in initiating structure tasks (e.g. behaviors aimed at the completion of a task in a effective and efficient manner). This gender effect is due to the fact that certain tasks are considered stereotypically feminine while others stereotypically masculine.

3.13 Internal constraints: affect, self-schemas and level of identification

As Lord et al. (2001) note "[...] constraints may also arise from *characteristics of the followers*, which operate through values, norms, affect, or goals to influence the leadership prototype that is generated". The internal constraints identified by the authors are: affect, self-schemas and self-identity level.

The affective tone[30] of a perceiver is likely to have a strong influence in perception of leadership, since affective processing is fast and occurs early in the handling of any social stimulus. The tendency to "like or dislike" a person may produce a strong internal constraint on subsequent perceptions of leadership and on the effectiveness of leader's behaviors.

Self-schemas are images of the self, different for every individual. When these organized collections of information are chronic, or generally accessible (Lord et al. 2001) it is reasonable to

[30] As taken from "The Medical Dictionary" available at http://www.books.md/index.html "Affective Tone" is "The mental state (pleasure, repugnance, etc) that accompanies every act or thought". Accessed 27 February 2004.

maintain that they can substantially affect the prototype generation. As Lord et al. (2001) put as an example that might particularly fit the open-source environment " [...] an individual who sees herself as being dependent may be more likely to activate a leadership prototype in interpreting the behavior of other individuals with whom she is interacting than an individual who sees herself as being independent." (Lord et al., 2001).

Another aspect of the self, is the level at which self-identity is defined. Lord et al. maintain that self-identities can be defined at an individual, interpersonal or collective level, and that only one level is operative at a time (Lord, Brown, and Freiberg, 1999). The activated aspect of the self-identity, primed by the context, is called "Working Self-Concept" (Markus and Wurf, 1987). In particular, when the self-identity is defined at a group level, traits and behaviors oriented to the interest and identity of the whole group (e.g. self-sacrificing and cooperative) are likely to have a strong influence in the prototype generation. Accordingly, it seems reasonable to assume that different open-source developers may be characterized by different levels of group identification. Kenneth Christiansen, stressed during an interview his orientation towards the product (i.e. The GNOME Desktop Environment) and the group:

"[referring to The GNOME Project] ... it's like we are a big family with shared grand goals. Even though I do not like to maintain some modules because they are boring to me – I still do that for the sake of GNOME!" [Interview with Kenneth Rohde Christiansen, 8 Oct. 2003].

On the other hand, participants whose self is defined at a more individual level[31] might be less likely to hold a prototype of a project leader based on behaviors oriented to the maintenance of the group identity.

[31] They might be those "external" participants that submit only a few patches and who are not involved in the project life as such.

Part II: Research objective and analysis

4 Research objective and structure

The purposes of this chapter are:

- Present the preliminary interviews that inspired the definition of the research objective.
- Outline the research objective and the theoretical approach used.
- Briefly describe the research path I followed to achieve the research objective.

4.1 Preliminary interviews behind the definition of the research objective

During the early stage of the research process of this thesis, two interviews have been conducted with Danish coders involved in development of open-source software. The interviews consisted in a set of 36 questions related to the topic of leadership in the open-source environment[32]. The objective was to find some interesting subjects on which to develop this paper. The questions posed in the interviews ranged from the description of a project leader’s tasks, to the process of leaders emergence and conflict resolution.

Answers related to the question “What are the characteristics of a good project leader?” provided interesting clues that were worth being investigated further. The respondents’ answers suggested a clear idea concerning the attributes and behaviors of a “good open-source project leader”. The interviewees’ typical description spun around some recurrent concepts related to project leader’s perceived knowledge and capacity to maintain friendly and supportive relations with other project contributors.

Departing from this early observation, I wondered what were the main factors making up the contributors’ idea of a “good open-source project leader”.

4.2 Research objective

The objective of this thesis is to identify the main factors describing the contributors’ personal believes concerning the characteristics of a “good open-source project leader”. Even though numerous attributes and behaviors may describe the contributors’ idea of a “good leader”, some of them can be summarized by a small amount of main concepts called “factors”. If such factors are identified, the investigation and communication of the characteristics of a “good leader” can be performed in a more efficient way than considering single attributes and behaviors on their own.

[32] Appendix A contains the questionnaire of the interviews.

4.3 Theoretical model

This thesis refers the theoretical model developed by Lord, Foti and Phillips (1982) and revised by Lord, Brown, Harvey and Hall (2001) on the implicit theories of leadership held by individuals (see chapters 2 and 3). This theory maintains that leadership is a topic to be studied primarily in the followers rather than in the leaders themselves. Leaders emerge over a group not for their innate characteristics, but to the extent to which they are classified as such by the followers. The process of leader classification is based on a set of personal attributes and behaviors called “prototype”, which resides in the minds of the followers. Whenever a match between the perceived characteristics of a person and the leader prototype occurs, a consistent classification follows.

Seizing on this model, this thesis investigates the prototype used by open-source contributors to classify “good project leader”. Specifically, this paper focuses on the identification of the main factors summarizing the contributors’ prototype.

4.4 Research structure

Three chapters are dedicated to the study of the prototype of a “good project leader”.

Chapter five deals with the identification of the prototype content. The term “prototype content” refers to the attributes and behaviors contributors deem appropriate for a “good open-source project leader”.

Chapter six deals with the identification of the prototype structure. The term “prototype structure” refers to the main factors describing the attributes and behaviors of “good project leader”.

Chapter seven and eight present respectively a discussion of the identified factors and conclusions.

5 Identification of the prototype content

The purposes of this chapter are to:

- Identify the attributes and behaviors making up the contributors’ prototype of a “good project leader”.
- Classify the identified attributes and behaviors.

5.1 What is a “prototype content”?

The aim of this chapter is to identify the content of the prototype of a “good open-source project leader”. The term “prototype content” refers to the attributes and behaviors[33] making up the contributors’ idea of a “good leader”.

5.2 Method used to determine the prototype content

The approach used to determine the prototype content has been based on the effects prototypicality[34] has on the probability of item recall. Behaviors, which represent well the prototype of a person are recalled frequently by respondents. On the other hand, behaviors which do not fit the prototype tend to be recalled less frequently (see section 3.6).

For this purpose, an open-ended questionnaire has been administered to some developers engaged in the development of open-source software. Respondents have been asked to recall and write into a web-form as many characteristics of a “good project leader” as they could (Appendix B contains the complete questionnaire).

5.3 Launching the questionnaire

Data has been collected administering the questionnaire to 34 volunteers who accepted to participate in the research between the 2 December 2003 and the 9 January 2004. Personal email invitations have been sent to 67 randomly chosen open-source contributors, whose contacts have been taken from public mailing-list postings. No more than two or three contact emails were taken from the same mailing-list of a project. Thirty-four respondents agreed to participate[35]. The

[33] From now on, the term “behaviors” will be used as a shortened version of “attributes and behaviors”.
[34] Prototypicality measures how well a behavior represents an individual’s image of a certain person (see section 3.4).
[35] The participation rate was approximately equal to 51%.

invitation contained a link to a web-based questionnaire set up in the server facilities of the university. As a whole, 392 raw items have been collected[36].

5.4 Classification of the collected items

The first problem concerning the raw data was its classification and normalization. Each of the 34 submissions contained some redundant items, which required treatment. In addition, while some items were expressed in short terms e.g. “competent”, some others consisted in little but articulated behavioral descriptions e.g. “being a visible leader in the mailing list and other form of communications means”. Indeed, raw data had to be refined to get a clear picture of the prototype content.

The process of data refinement proceeded in the following manner:

1. The behaviors of the first submission inspired the determination of some initial “behavioral classes” necessary for item categorization.
2. The behaviors collected with the second submission have been categorized using the existing behavioral classes cited in point 1.
3. If some behaviors of the second submission did not fit the existing classifications, then new classes were extracted and kept apart for reuse.
4. The same process of categorization has been performed for all the subsequent contributions.

The labels for each behavioral class have been determined by considering the typical descriptions submitted by respondents. As an example, “patience” was an attribute frequently recalled by contributors. Even though some respondents submitted the term “patient” without any additional specification, the majority of those leaving a behavioral description about the same attribute reported “patience in dealing with the contributors”. In such case, the explicit description of an attribute was selected as a basis for the behavioral label.

The process of item classification has been performed with a certain level of approximation. Typically, approximation has been based on the similarity of the terms used e.g. “friendly” has been classified as “He is friendly and approachable” and concepts expressed e.g. “gives voice to the project” has been classified as “He acts as the spokesman of the project”.

[36] This value has been calculated consolidating all the results and counting the number of carriage returns (“return” to a new line character), excluding blank lines.

Whenever item classification was doubtful due to the level of abstraction of the terms or descriptions used, approximation has been performed after having contacted by email the original submitter for a clarification e.g. “leads by doing” after a clarification turned to be “He is active in the development of code”. Since respondents sometimes tended to clarify one attribute using two or more behavioral descriptions, all of them were used for the item classification e.g. “provides leadership to less experienced developers” after clarification has been classified either as “He is willing to mentor others” and ”He sets out the overall direction of the project”. Table 2 is an example of raw submission with the correspondent classification:

Raw submission	**Classification**		**Raw submission**	**Classification**
accept comments and suggestions	*He accepts comments and suggestions*		develops and develops	*He is active in the development of code*
gives voice to the group	*He acts as the spokesperson of the project*		good vision	*He has a vision, a future perspective for the project*
solve disputes	*He cares about settling conflicts between developers*		help with fixing bugs	*He helps contributors resolve technical problems*
constantly reviews code for quality testability	*He is committed to a quality product (he makes sure bugs are corrected)*		never nervous	*He is calm and equilibrate*
keeps deadlines	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*		friendly	*He is friendly and approachable*
maintains the project actively	*He is dedicated to the project (time, effort [..])*		knows every technical aspect of the project	*He is knowledgable on technical aspects of the project*
patient	*He is patient when dealing with contributors*		educate new developers	*He is willing to mentor others*
visible in the mailing list	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*		able to write real code	*He is technically competent - he writes real good code*
rigorous	*He is technically accurate*		listen to contributors	*He listens to what contributors have to say*
good motivation skills	*He motivates contributors*		review patches	*He reviews the work of others*
points out possible corrections	*He offers new approaches to problems*		creates infrastructure	*He sets up / maintains an infrastructure (cvs/website/mail-list)*

Table 2 -Example of a raw submission with related classification.

The complete list of raw items with corresponding classification is presented in Appendix D.

As the process of classification went on, the extracted behavioral classes were lumped on the group of available labels. At the end of the process of classification, the number of behavioral classes was equal to 89.

5.5 Results of the classification process

Once classified, absolute frequencies have been calculated for each behavioral class, counting the number of times it has been used in the process of item categorization. Behavioral classes used more than once in the classification of the behaviors submitted by the same respondent, accounted for one unit in the calculation of the frequencies. Table 3 presents a sample of such items, sorted by frequency of use in the classification process. Appendix C contains the full list.

Item #	Personal attributes and behaviors used more than 5 times for the classification of the raw items	Absolute frequency
1	*He is technically competent - he writes real good code*	20
2	*He has a vision, a future perspective for the project*	12
3	*He is dedicated to the project (time, effort [..])*	12
4	*He is committed to a quality product (he makes sure bugs are corrected)*	11
5	*He is easy to understand*	10
6	*He is patient when dealing with contributors*	10
7	*He helps contributors resolve technical problems*	9
8	*He is active in the development of code*	9
9	*He is delegative - he doesn't try to do all the work on his own*	9
10	*He is friendly and approachable*	9
11	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	9
12	*He is knowledgeable on technical aspects of the project*	8
13	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	8
14	*He establishes the roadmap/plan for the development*	8
15	*He keeps everyone informed on the project directions and achievements*	7
16	*He offers new approaches to problems*	7
17	*He accepts comments and suggestions*	7
18	*He is willing to mentor others*	7
19	*He makes decisions - public debate could drag ineffectively for weeks*	7
20	*He coordinates the efforts of the contributors (e.g to avoid overlapping contribution and to make things work together)*	6
21	*He encourages participation to the project*	6
22	*He is an experienced coder*	6
23	*He praises developers for their contribution*	6
24	*He listens to what contributors have to say*	6
25	*He sets out the overall direction of the project*	6
26	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	6
27	*He answers emails in a timely fashion*	6
28	*He cares about settling conflicts between developers*	5
29	*He decides what code/features to include or not to include*	5
30	*He discusses in the open various approaches with other coders*	5
31	*He is polite and respectful*	5
32	*He is concerned with getting the things done*	5
33	*He makes sure that guidelines and coding procedures are available to developers*	5
34	*He accepts external contributions (he is willing to say "that piece of work is better than mine")*	5
35	*He acts as the spokesperson of the project*	5
36	*He represents the project in conferences and the like*	5

Table 3 – Behavioral classes used more than 5 times in the classification of the raw items.

6 Identification of the prototype structure

The purposes of this chapter are to:

- Explain what the term “prototype structure” means and why it is worth investigating.
- Describe the second questionnaire used to collect the necessary data to study the prototype structure.
- Describe the application of factor analysis to identify the prototype structure.

6.1 What is a “prototype structure” and why is it worth investigating?

The aim of this chapter is to identify the structure of the prototype of a “good open-source project leader”. The term “structure” means the basic concepts that summarize developers’ idea of “good leadership”.

The contributors’ image of a “good leader” has proven to be composed by numerous attributes and behaviors[37]. Nonetheless, it is possible to identify some basic behavioral descriptions that summarize this prototype. In particular, some behaviors may be grouped together by some latent relationship, in the mind of the respondents. If such relationship can be identified, it can be used to summarize the personal believes of the open-source contributors.

A statistical approach called “factor analysis” is used to identify latent relationships between variables describing a phenomenon. Such analysis allows a researcher to classify the correlated variables into groups called “factors”. Accordingly, in this chapter such tool will be applied to the available data.

However, factor analysis cannot be applied directly to the list of items classified in chapter 5. This plain list does not provide the necessary information to determine how behaviors tend to group in the minds of the contributors. Therefore, a method to identify such clusters of behaviors had to be figured out for the second stage.

The methodology used by psychological studies consists in administering a questionnaire of behaviors which asks respondents to rate each item dependently on how well it describes a person they have classified under a certain category. Subsequently, factor analysis figures out any latent relationship among behaviors by analyzing the way ratings tend to co vary. If the analysis identifies any regularity in the way behaviors are rated, a factor is said to exist.

[37] The study of the prototype content described in chapter 5, identified 89 attributes and behaviors.

In the same manner, a rating-based questionnaire which asked respondents to describe a real life “good project leader” has been put together. The next section will describe the principle behind its creation.

6.2 The principle behind the rating-based questionnaire

The principle on which the rating-based questionnaire has been created, refers to the effect the prototypes have on the way individuals are described. As noted in section 3.7, people tend to describe a person accordingly to the prototype they have used for their classification (Eden and Leviatan, 1975; Rush, Thomas and Lord, 1977). This means that the actual behavior of a person is relatively unimportant for behavioral descriptions, since descriptions are mainly based on the prototype we use to classify a target person.

Accordingly, in the questionnaire set up for this stage, participants have been asked firstly to classify the leaders they dealt with dependently on their level of satisfaction and secondly, to describe the classified leaders by rating each listed behaviors on a “1 to 9” scale. The next section will describe the choices of leader classifications available to the respondents.

6.3 Leaders’ classification accordingly to the respondents’ level of satisfaction

If respondents were only asked to describe their leaders, there would be no way to verify in which category such leaders were classified – specifically whether they were “good leaders” or “bad leaders”. For this reason respondents have been asked to classify the leader they dealt with before performing the description.

However, respondents have not been asked to classify their leader using categories like “good project leader” or “bad leader”, since such qualifications might have been perceived as too personal or just inappropriate. Therefore, a different approach has been selected. Respondents have been asked to classify their leaders according to the level of satisfaction they experienced while working with them.

Five evaluative statements were available for leader classification to provide respondents with sufficient flexibility when expressing their level of satisfaction. The evaluative statements were:

- “I am VERY satisfied with my project leader”
- “I am satisfied with my project leader”
- “I am sufficiently satisfied with my project leader”
- “I am unsatisfied with my project leader”
- “ I am NOT satisfied at all with my project leader”

However, the main focus in this thesis is on leaders that contributors are satisfied with. Hence, only the submissions of the contributors that stated to be generally satisfied with their leader, have been considered for the analysis. Specifically, the submissions relative to the first three evaluations have been grouped and considered as equivalent.

After having performed the classification, respondents have been asked to describe the classified leader by rating each behavior listed in the questionnaire. The next section shows which behaviors have been chosen to be part of the survey.

6.4 Selecting the attributes and behaviors to build the questionnaire

In order to focus the study on the most relevant behaviors of a “good project leader”, 36 out of 89 identified items have been retained for the rating-based questionnaire. The behaviors selected were the ones used at least five times in the classification process carried out in the first stage.

Indeed, all the 89 behaviors are part of the prototype of a “good project leader”. However, each behavior has a different “weight” in the classification process, which can be approximated by the number of times it has been recalled by contributors[38]. Behaviors recalled more frequently are those that capture the essence of the prototype and hence are most important in classifying a “good leader”.

Before converging into the questionnaire, the picked items have been randomly mixed to eliminate any possible bias due to the order they were shown in. Appendix E contains all the snapshots of the questionnaire.

[38] See section 3.6 for the effect prototypicality has on probability of item recall.

6.5 Launching the questionnaire

For this questionnaire, invitations have been publicly posted between the 2 February 2004 and the 22 February 2004 in the development mailing lists of the open-source projects listed in Appendix F. The invitation posted contained a link to the questionnaire web page. Access to the questionnaire was not regulated by any PIN code or any other authentication method[39] to facilitate questionnaire participation.

As a whole, 104 valid submissions[40] have been collected, among which:

- 36 respondents stated to be "VERY satisfied" with the leader of the project in which they were active the most;
- 35 respondents stated to be "satisfied"
- 27 stated to be "sufficiently satisfied"
- 6 respondents stated to be "unsatisfied"
- No respondents said to be "NOT satisfied at all"

6.6 Performing factor analysis on the data

In the rating-based questionnaire respondents were asked to describe their project leaders by rating each behavior listed on a scale from 1 to 9. We can wonder whether the ratings collected for each behavior are moving independently from one another or if some of them tend to move together in a consistent manner - thus revealing that a latent relationship connects them.

Factor analysis is a statistical approach capable of figuring out any latent relationship among variables. Accordingly, "principal axis factoring"[41] has been applied to the available data, using a statistical software application called "SPSS version 12"[42].

[39] The fact that the questionnaire access was not controlled by any authentication method (e.g. PIN code), questions the quality of the submissions. As a way to ensure a reasonable level of submissions quality, the questionnaire has been publicized ONLY on relevant development mailing list (advocacy mailing list have been carefully avoided). In addition, those contributions that missed more than 1/3 of the 36 questions have been eliminated.
[40] The total amount of raw submission was equal to 198 out of which 94 had to be eliminated because either totally left blank or not reaching the minimum number of answers for contribution validity (24 items out of 36). Hence the response rate was equal approximately to 47%.
[41] "Principal axis factoring" is the exact label for the factor analysis I introduced.
[42] In this chapter factor analysis is considered as a useful tool to achieve the objective of this thesis, without going into an in depth description. However, an exhaustive explanation of factor analysis is presented by the article of Rudolph J. Rummel, "Understanding Factor Analysis" available at: http://www.hawaii.edu/powerkills/UFA.HTM

It is important to outline two characteristics in the way this type of tool extract factors from a set of data:

1. Factors extracted are uncorrelated (i.e. it is possible to think of them as perpendicular lines in a space.)
2. Factors are extracted in a specific order, from one that explains the greatest regularity in the way data tend to move, to those that marginally account for lower levels of such regularity.

From a researcher’s point of view, not all the identifiable factors are equally interesting. As said, factors tend to explain less and less amounts of latent regularity in the data as they are extracted. Thus, the first question in a factor analysis is to decide how many factors to extract, knowing that factors that are furthest away can only explain lower level of regularity.

The rule of thumb for determining the number of factors to extract is to retain those associated to an “eigenvalue greater than one”, to say those factors that explain at least the variance accounted for by one of the original variables. Following, is a graphic representation of the eigenvalues associated to each factor, called “scree plot”:

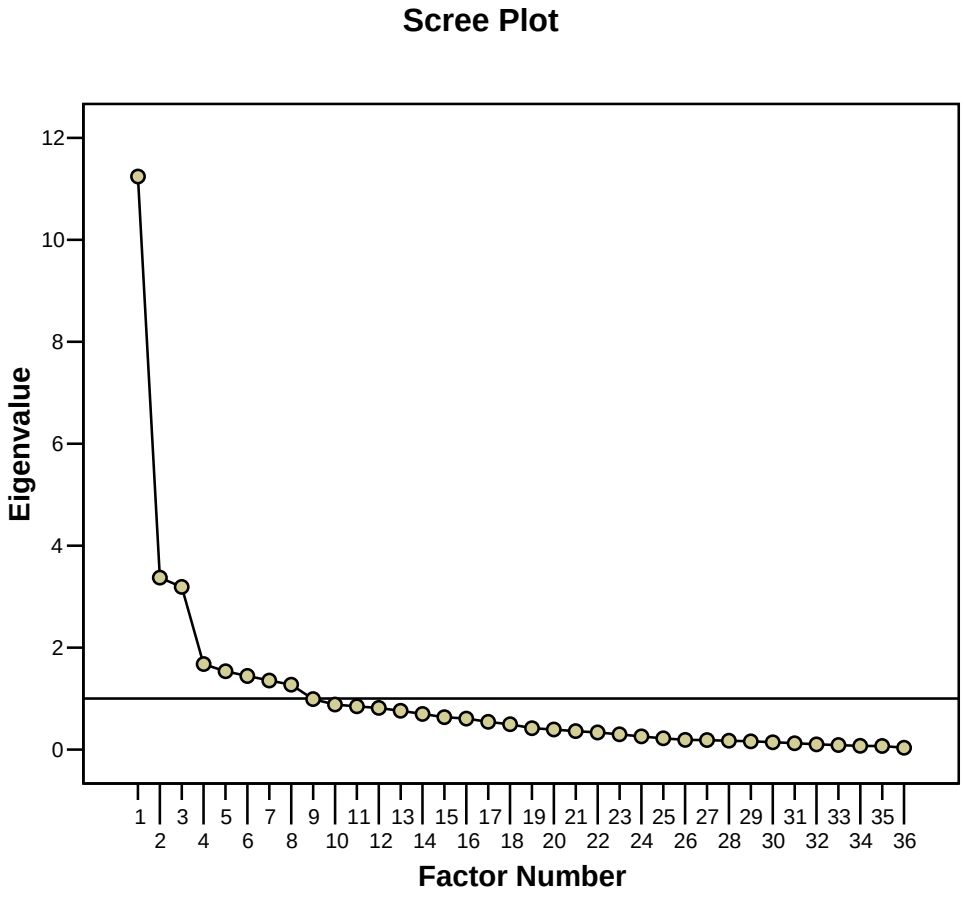


Figure 2 - Graphical representation of the eigenvalues associated with each factor

Figure 2 shows each factor associated to the amount of variability it can explain. As it can be seen from the scree plot, the first factor account for the largest amount of regularity in the data. Accordingly, all the following factors explain decreasing levels of regularity.

From the scree plot in Figure 2, eight factors are associated to an eigenvalue greater than one and thus are worth to be extracted. However, since the first three factors account for the majority of the total variance, those are the ones that will be considered in this analysis.

The first thing to do after the selection of the number of factors to extract, is to estimate how much variance they account for, as a set. Such information is contained in Table 4:

Factor	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared Loadings		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	11.242	31.227	31.227	10.734	29.818	29.818	7.311	20.308	20.308
2	3.372	9.367	40.594	2.912	8.088	37.906	5.574	15.483	35.791
3	3.190	8.861	49.455	2.661	7.393	45.298	3.423	9.508	45.298
4	1.677	4.658	54.113						
5	1.536	4.266	58.379						
6	1.444	4.010	62.389						
7	1.355	3.763	66.152						
8	1.273	3.535	69.687						

Table 4 - Total variance explained by the extracted factors

The values grayed out in Table 4 show that the three first factors account for the 45% of the total variance. This means that the 45% of the variation of the 36 variables can be explained by three latent relationships (factors).

Now that we know that three relationships exist, the question is: "What are the behaviors involved in such relationships?"

6.7 Identification of the behaviors making up each factor

Factor analysis produces an "unrotated factor matrix"[43]. Such matrix shows how many factors have been extracted and the amount of variability explained by each one. However, this matrix does not provide an easy way to identify the behaviors involved in each relationship. For this purpose, a *rotation* must be applied to the unrotated factor matrix.

To understand what a rotation is, we can think of factors as perpendicular axis projected in a space in which behaviors are plotted as points. Such points may tend to form distinct clusters in the space, thus revealing that they are kept together by some latent relationship. Rotations "rotate" factor axis until they identify such clusters and the behaviors that make them up.

For this part of the analysis, a Varimax orthogonal rotation has been applied to the unrotated matrix shown in Appendix G. Such orthogonal rotation has been selected because it permits to identify

[43] This matrix is shown in Appendix G

clusters of behaviors that are uncorrelated. This means that the identified behaviors represent the “essence” of the phenomenon and thus do not contain any redundant information.

The output of such rotation is shown in the “rotated factor matrix” in Table 5:

	Factor		
	1	2	3
discusses_openly	.637	.108	.281
accepts_contributions	.632	.003	.273
patient	.763	.214	.080
answers_emails_promptly	.594	.393	-.109
easy_to_understand	.705	.253	-.091
delegative	.557	.103	.198
settle_conflicts	.588	.339	-.094
friendly	.793	-.060	.060
keeps_informed	.465	.501	.102
listens	.803	.305	-.003
accepts_suggestions	.724	.228	.130
praises_developers	.493	.459	.006
polite	.718	.130	.105
represents_the_project	.114	.513	-.120
talks_for_the_group	.248	.453	-.191
knowleadgeable	.251	.089	.684
experienced	-.006	-.071	.875
competent	.215	-.058	.749
sets_direction	.120	.570	.297
has_vision	.347	.594	.132
mentor_others	.339	.514	-.066
helps_contributors	.422	.365	.372
offers_new_approaches	.393	.372	.363
decides_what_to_include	-.220	.312	.478
makes_decisions	-.020	.710	.073
maintains_infrastructure	.107	.421	.107
establishes_roadmap	.102	.701	.251
coordinates_work	.505	.344	.100
makes_guidelines	.208	.450	-.001
commited_to_quality	.164	.370	.395
encourgaes_participation	.582	.431	.262
dedicated	-.022	.472	.098
get_things_done	.265	.648	.084
emphasizes_deadlines	.235	.475	-.006
present	.118	.243	.122
active	.065	.096	.594

Table 5 - Rotated factor matrix of the three factors (Varimax)

The rows of such matrix define the behaviors[44] while the columns define the extracted factors. The cells of the matrix show “factor loadings” that indicate how much each behavior correlates to each factor - basically how near a point is to the rotated axis. Such factor loadings range from 1 to -1. While a factor loading of 1 indicates a perfect positive correlation between a behavior and a factor (i.e. the behavior lies exactly on the factor), a factor loading of 0 indicates the absence of correlation between a specific pair of behaviors and factor.

As a rule of thumb, items that correlate at least 0.4 on the highest factor and less than 0.3 on the others are considered as part of a factor. Following the factor loadings marked accordingly to this rule in Table 5, it is possible to determine the behaviors part of each factor. These behaviors grouped for each factor, are shown in the following table:

1	2	3
discusses_openly	represents_the_project	knowleadgeable
accepts_contributions	talks_for_the_group	experienced
patient	sets_direction	competent
easy_to_understand	makes_decisions	active
delegative	maintains_infrastructure	
friendly	establishes_roadmap	
listens	makes_guidelines	
accepts_suggestions	dedicated	
polite	get_things_done	
	emphasizes_deadlines	

Table 6 - Behaviors part of each of the three factors extracted

Now that the items composing each factor have been identified, the question is: “What do these factors represent?”.

[44] The rows of the matrix shown in Table 5 reports stripped-down versions of the behavioral descriptions used in the rating-based questionnaire.

7 Discussion

The discussion will focus on three points:

- The interpretation of the factors.
- The relation between the factors.
- The capacity of the three factors to describe a “good project leader”.

7.1 The interpretation of the factors

The analysis performed in the precedent chapter revealed that three main factors summarize the developers’ prototype of a “good project leader”. However, such analysis does not tell us what those factors represent. This is the reason why the factors must be interpreted.

The factor interpretation is performed by considering all the behaviors making up each factor. As shown in Table 6 the behaviors that correlate strongly with the first factor depict a friendly and patient project leader, willing to listen to the contributors’ suggestions and delegate tasks. This leader is easy to understand by the contributors, he publicly discusses different technical approaches and does not refuse to recognize the good work of others. This description identifies a category of behaviors that stresses the importance of the consideration for the person. Therefore, this factor has been labeled “person orientation”.

The second factor is composed by behaviors that emphasize the leader orientation towards the task and the facilitation of work. This leader is perceived as dedicated to the project and goal-oriented. He takes care of planning activities, makes sure that guidelines are available for contributors, and emphasizes deadlines for the submission of new software functionalities. Such leader performs the necessary organizational activities for the project to keep it in process. Accordingly, this cluster of behaviors has been labeled “task orientation”.

The last group is made up by a few items which nonetheless correlate strongly with the factor. The technical skills of the leader, his level of experience and the involvement in the production of code depict a leader that “knows and does”. This factor has been labeled “competence/activity”.
The three interpreted factors are shown in the following table:

Person orientation	Task orientation	Competence/Activity
discusses_openly	represents_the_project	knowleadgeable
accepts_contributions	talks_for_the_group	experienced
patient	sets_direction	competent
easy_to_understand	makes_decisions	active
delegative	maintains_infrastructure	
friendly	establishes_roadmap	
listens	makes_guidelines	
accepts_suggestions	dedicated	
polite	get_things_done	
	emphasizes_deadlines	

These three factors summarize the contributors’ expectations towards their project leader. A “good leader” is expected to be a competent programmer active in the development of code, who engages in the provision of organizational services instrumental to the achievement of the project goals and able to maintain friendly and considerate relation with the contributors.

The process of interpretation permits the identification of the main concepts behind the contributors’ idea of a “good leader”, nevertheless it does not tell us whether there is any relation between these factors. The next section will answer this question.

7.2 The relation between factors

If we go through Table 5, we notice that some behaviors correlate highly on two or three factors. As an example, “He settles conflicts between developers” and “He praises developers for their contributions” are both highly correlated with the “person orientation” and “task orientation” groups. The same can be said in respect to “He coordinates the efforts of the contributors” and “He keeps everyone informed on the project directions and achievements”. Why do such behaviors correlate highly on different factors?

My explanation is that, in the open-source environment it can be difficult to draw a clear distinction between task oriented and person oriented behaviors, for two reasons:

1. Some behaviors oriented to the facilitation of the work and the provision of organizational services, are perceived by the community also as expressions of person consideration. As an example, “coordinating the efforts of the contributors” is not only perceived as a way to

achieve the project goals, but also as a sign of “kindness and friendliness”, since the project leader is typically not paid for that.

2. Some behaviors oriented to the consideration of the person and the maintenance of group well-being, are necessary for the achievement of the project goals. This is especially true in the open-source environment in which contributions are unpaid. “Praising developers for their contributions” is not only a sign of person consideration, but also a way to keep the flow of unpaid contributions alive. On the other hand, “Settling conflicts between developers” is necessary to avoid the fragmentation of the group and keep the project in process.

This observation reveals that, task oriented and person oriented behaviors are intertwined. However, it is difficult to say to what extent such factors are overlapping, since not all the behaviors correlate highly on both factors. As an example “He is friendly and approachable”, correlate highly with the person orientation and low with task orientation. This means that being friendly is a good sign of person orientation, but not necessarily of task orientation.

On the other hand, the behaviors shown in Table 5 do not tend to intercorrelate very much with the competence/activity factor, meaning that this group is distinct from the task and person orientation ones.

7.3 The capacity of the three factors to describe a “good project leader”

So far, three factors have been identified and interpreted, which explain the 45% of the total variation of the data. This is a remarkable amount but there is still a portion of variation that is not accounted for by any of the three factors. Therefore, what explains the rest of the variation?

Other factors than the three extracted ones might explain the remaining portion of variability. The scree plot shown in Figure 2 suggests the existence of five additional relationships between behaviors. However, these factors cannot account for all the variation that is left[45].

[45] The purpose of this thesis is to focus of the main latent relationship in the data. This is the reason why, such additional factors have not been extracted.

The remaining variation can be due to the absence of any regularity in the way “good project leaders” have been described by respondents. Now the question is “what is the source of such irregularity?”

A possible explanation can be related to the context surrounding the project. As Lord et al. (2001) suggest, there are different contextual constraints that shape the prototype of a leader (see section 3.11). National culture of the followers, the nature of the task a leader is expected to perform, followers self-schemas, are all forces that make the prototype of a “good leader” variable from situation to situation. Among these, I think that the nature of the task a leader is expected to perform affected substantially the results of the analysis. This is especially true if we recognize that leadership often is shared among open-source contributors by distributing responsibilities for different tasks (see section 1.8). As an example, in the FreeBSD project a “core team” assigns maintainers to different areas of code and solve disputes between them, while a “release team” is expected to emphasize deadlines for the submission of new software functionalities (Saers, 2003). This observation is particularly appropriate if we consider the mailing lists of the projects in which the invitation to the questionnaire has been posted. Most of them are big projects, likely to present complex leadership arrangements, in which responsibilities are spread among a large number of participants. In such cases, contributors may hold different prototypes of “good leaders” depending on the tasks leaders are expected to perform.

This thesis has studied the prototype of a “good project leader” in general terms, without taking into account the leadership arrangements of a project. However, if information concerning the distribution of the responsibilities within projects had been collected, there most probably would have emerged a clearer picture of the prototype of a “good leader”.

8 Conclusions

The objective of this thesis was to identify the main factors describing the open-source contributors’ personal believes concerning the attributes and behaviors of a “good project leader”. The contributors’ believes proved to be rooted into three factors, which represent:

- The project leader’s orientation towards the maintainance of friendly and considerate relationships with the contributors (person orientation).

- The project leader’s orientation towards the achievement of the project goals, through the provision of organizational services to the community (task orientation).

- The project leader’s competence and activity in the development of code (competence/activity).

Specifically, the task and person orientation factors are not clearly distinct in the minds of the contributors. The contributors’ believes concerning “good leadership” interwine the leader’s orientation towards the facilitation of the work and achievement of the project goals with his orientation towards the maintainance of the relations with the developers. There are two possible explanations for this effect:

1. Open-source contributors consider the leader’s provision of organizational services to the community as a sign of concern towards getting the work done (task orientation) and also as a sign of kindness and friendliness (person consideration), since the leader typically performs such role for free.

2. Open-source contributors perceive that the achievement of the project goals cannot be performed without keeping considerate relationships with developers and maintaining the social well-being of the community.

In addition, this thesis shows that the expectations held by contributors towards “good leaders” are variable. Even though different contextual forces may constrain the contributors’ expectations, the leadership arrangements of the project (i.e. how responsibilities are shared among participants) play

a strong role in shaping the contributors’ personal believes concerning the characteristics of a “good project leader”.

Appendix A – Preliminary interview questions

1. What is your personal conception of the term “leadership”? Can you state your personal definition?

2. Can you lower the concept of “leadership” in the open source environment? What does the term “he/she is the leader of this project” or “they are the leaders of this project” mean?

3. Can you cite any example of a leader (person or group) of an open-source project?

4. Why do you think of him/them as leader/s?

5. What is the difference between a “leader” and a “maintainer”?

6. Can you think about maintainers being leaders and maintainers NOT being leaders? What does justify this existent or missing overlapping?

7. Can you cite any leaderless open-source project? What makes you think that there is no leader in such projects?

8. Can you explain why some open-source projects do not have a leader?

9. Why do some projects present a shared-leadership model, while others a single person leader?

10. Into those projects showing a clear leadership dynamic, is the leader occupying a formal position?

11. What are the characteristics of a leader candidate?

12. Is there any formal procedure governing the “election” of the leader?

13. Is there any informal path through which contributors become leaders? If any, can you describe it?

14. To what extent do you think the role of a “leader” is a natural emergent figure opposed to the result of an appointment procedure?

15. Is the emergence of a leader related to any condition you can state?

16. Can you describe the difference between a leaderless and a leader-endowed project? In which way you think they perform differently?

17. How important is the role of a leader for the success of an open-source project? Can you state the reasons?

18. What are the tasks carried out by a leader? What type of tasks are exclusively carried out by the leader or the leading group?

19. To what extent do you think the task of a leader is strictly “technical”? (e.g. just a gatekeeper to the CVS)

20. Can you cite any example project that owns a lot of its success to the organizational capacities of the leader? What is it special with him? What kind of tasks does he perform?

21. What are the characteristics of a good leader? What are the prerequisite for a good leader?

22. What shouldn’t a leader do? Can you state any “don’t” for an open-source leader?

23. Can you state any open-source project in which, even existing a formal leader, still there are some persons exerting a considerable influence on the project? Along what dimensions is this influence exerted? What makes this influence effective/ineffective?

24. Have you ever felt being an “example” for your members? In relation to what aspects? Why contributors were looking at you as an “example”?

25. Have you ever felt like leading the group, or part of it? What makes you think you were occupying a leadership position?

26. Let us say you had to cite the most representative contributor of the project you are participating to. At which developer would you point at? Why do you think of him as the most representative contributor of the project? Do you think he exerts any influence over the other members? If yes, in respect to what aspects?

27. What are the “sources of power” backing the capacity of a leader to influence contributors?

28. How would you go if you had to “persuade” a contributor to perform a certain task? Have you ever been effective doing this? What made your “persuasion attempt” successful?

29. Do you think there are tasks for which it is easier to “persuade” contributors to perform them?

30. Have much of your work, requiring a sort of interaction with others, is resolved through “orders”?

31. Have you ever felt authoritative? Can you cite any example of leader authoritative behaviour in an open-source project? What did justify his authority?

32. What is the nature of the most common conflicts in the open-source environment? How are such conflicts resolved?

33. Can you remember any situation in which a third person intervened resolving a conflict? What was the source of his capacity to settle such conflict?

34. Do you ever intervene in resolving any disputes? Why have you felt like doing that? Have you been effective for this purpose? What did it make you effective?

35. Is there anyone able to assign tasks? How successful is he at that? Has he any authority doing that? What does justify his authority? If he has not any authority, what makes him effective?

36. How would you identify the leader (or leaders) in an open-source project?

Appendix B – First open-ended questionnaire

QuizComposer©
©2001- Olaf Kayser & Gunnar Mohr
Contact: admin@quizcomposer.dk
Main site: www.quizcomposer.dk

Good leadership in the open-source community v0.3

Thank you for participating in this survey! The results of the following questionnaire will be used for my **master thesis** on leadership in the open-source environment. Your contribution will not be publicly identified in any way unless agreed by you.

You will have the chance to read a brief description of the objective of my project after having submitted the questionnaire. At the moment I will not give you any details, in order to avoid influencing your answers.

This questionnaire will require around 15 minutes to complete. Please answer the questions in the order they are presented. A plain text version of the following questionnaire is available here.

Thank you for participating,

Gianluca Bosco g.bosco@inwind.it

.

What's your age?

.

What's your nationality?

.

Look into your past experience and list *traits, attributes and behaviors* of a good open-source project leader. Please, generate as many items as you can (indicatively, a set of 15 items is sufficient)

Separate each item (single word or brief description) by using commas:

.

For how long have you been contributing to open-source projects? (e.g. 3 months; 10 years;)

.

On an average basis, how many *hours a week* do you spend contributing to open-source projects? (e.g. 15)

.

How many project have you been contributing to, as an open-source developer? (e.g. 6)

.

State the names of the projects in which you spend the most time, in a decreasing order.

Separate each project name by using commas:

.

Have you ever been an open-source project leader?

- ○ Yes
- ○ No

.

If you agree in being contacted for eventual clarifications or extensions of your submission, please insert a contact address (email/ICQ/IRC/telephone), otherwise leave the field blank.

.

Do you wish your identification details to be disclosed?

- ○ Yes
- ○ No

.

Please, insert your name or nickname:

.

If you like, you can use the following field to post comments concerning this questionnaire. For example things you did not like, vague questions, etc.

This was the last question. After having submitted the questionnaire you will have the chance to read a description of my project.

When ready click checkbox and then button: ☐ SUBMIT THIS FORM

Author: Gianluca Bosco
Thesis's supervisor: Ph.D. Kasper Edwards
Technical University of Denmark - Department of Manufacturing Engineering and Management

Appendix C – List of the extracted behavioral classes

Item #	Extracted behavioral classes (1 of 2)	Absolute frequency
1	*He is technically competent - he writes real good code*	20
2	*He has a vision, a future perspective for the project*	12
3	*He is dedicated to the project (time, effort [..])*	12
4	*He is committed to a quality product (he makes sure bugs are corrected)*	11
5	*He is easy to understand*	10
6	*He is patient when dealing with contributors*	10
7	*He helps contributors resolve technical problems*	9
8	*He is active in the development of code*	9
9	*He is delegative - he doesn't try to do all the work on his own*	9
10	*He is friendly and approachable*	9
11	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	9
12	*He is knowledgeable on technical aspects of the project*	8
13	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	8
14	*He establishes the roadmap/plan for the development*	8
15	*He keeps everyone informed on the project directions and achievements*	7
16	*He offers new approaches to problems*	7
17	*He accepts comments and suggestions*	7
18	*He is willing to mentor others*	7
19	*He makes decisions - public debate could drag ineffectively for weeks*	7
20	*He coordinates the efforts of the contributors (e.g to avoid overlapping contribution and to make things work together)*	6
21	*He encourages participation to the project*	6
22	*He is an experienced coder*	6
23	*He praises developers for their contribution*	6
24	*He listens to what contributors have to say*	6
25	*He sets out the overall direction of the project*	6
26	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	6
27	*He answers emails in a timely fashion*	6
28	*He cares about settling conflicts between developers*	5
29	*He decides what code/features to include or not to include*	5
30	*He discusses in the open various approaches with other coders*	5
31	*He is polite and respectful*	5
32	*He is concerned with getting the things done*	5
33	*He makes sure that guidelines and coding procedures are available to developers*	5
34	*He accepts external contributions (he is willing to say "that piece of work is better than mine")*	5
35	*He acts as the spokesperson of the project*	5
36	*He represents the project in conferences and the like*	5
37	*He makes sure that documentation of the software is available*	4
38	*His mind is open to new solutions*	4
39	*He promotes the project (advertises it, maintains PR etc)*	4
40	*He reviews the work of others*	4
41	*He has a mature sense of responsibilities*	4
42	*He recruits new project members*	4
43	*He organizes the project activities*	4
44	*He discourages heated arguments*	3
45	*He has a general overview of the project*	3

Item #	Extracted behavioral classes (2 of 2)	Absolute frequency
46	*He motivates contributors*	3
47	*He is calm and equilibrate*	3
48	*He is enthusiast of the project and excitable*	3
49	*He is intelligent*	3
50	*He is technically accurate*	3
51	*He believes in the open-source ideology*	2
52	*He builds personal relations with contributors*	2
53	*He gives feedbacks on the work of others*	2
54	*He explains publicly his choices*	2
55	*He has cultural sensitivity*	2
56	*He is collaborative*	2
57	*He is diplomatic*	2
58	*He is humble*	2
59	*He is humorous - development is fun with him*	2
60	*He is not easily drawnable into flamefests*	2
61	*He is optimist*	2
62	*He knows about cutting edge techniques*	2
63	*He takes care of building the final package*	2
64	*He throws project members out if they have negative overall contribution*	2
65	*He starts a project out of a desire to learn or expand the knowledge of a new technology*	2
66	*He is organized*	2
67	*He knows at least one second language*	2
68	*He makes no discriminations*	1
69	*He actively looks for people to work on things that nobody picks up*	1
70	*He adjusts to changes fast*	1
71	*He considers aesthetical issues*	1
72	*He divides the work among contributors*	1
73	*He does not rule out people when people don't fit his vision*	1
74	*He establishes partnership with companies and others projects*	1
75	*He has an independent character*	1
76	*He has been under a good project leader for a long time*	1
77	*He has the ability to think like a user (to approximate their needs)*	1
78	*He is a perfectionist*	1
79	*He persuades others that the chosen idea is best*	1
80	*He is always looking for improvements*	1
81	*He is knowledgeable on legal aspects*	1
82	*He is prone to step out when needed*	1
83	*He is realistic on what could be done and achieved*	1
84	*He is self-motivated*	1
85	*He is willing to prune to improve growth*	1
86	*He favors information exchanges*	1
87	*He is trustworthy*	1
88	*He keeps the project interesting and alive*	1
89	*He organize meetings (IRC, email [..])*	1

Appendix D – List of the raw items and related behavioral classification

1	*He accepts comments and suggestions*	accept comments and suggestions
2	*He accepts comments and suggestions*	be open to suggestions by others
3	*He accepts comments and suggestions*	take advices from the team
4	*He accepts comments and suggestions*	listens to developers comments
5	*He accepts comments and suggestions*	listens to criticism
6	*He accepts comments and suggestions / He accepts external contributions (he is willing to say "that piece of work is better than mine")*	encourages feedback and contributions
7	*He accepts comments and suggestions / He accepts external contributions (he is willing to say "that piece of work is better than mine")*	accepts feedback and contributions
8	*He accepts comments and suggestions / He makes decisions - public debate could drag ineffectively for weeks*	accepts different points of view but then he must take a decision
9	*He accepts external contributions (he is willing to say "that piece of work is better than mine")*	accept external contributions
10	*He accepts external contributions (he is willing to say "that piece of work is better than mine")*	be willing to accept code from others
11	*He accepts external contributions (he is willing to say "that piece of work is better than mine")*	accepts code
12	*He actively looks for people to work on things that nobody picks up*	activily looks for people to work on things that nobody picks up themselves
13	*He acts as the spokesperson of the project*	gives voice to the group
14	*He acts as the spokesperson of the project*	talks for the project
15	*He acts as the spokesperson of the project*	talk person
16	*He acts as the spokesperson of the project*	makes announcements for the project
17	*He acts as the spokesperson of the project*	project spokesperson
18	*He adjustes to changes fast*	Ability to ajust to changes FAST
19	*He answers emails in a timely fashion*	respond to emails: be available and responsive when other contributors (and potential contributors) ask questions
20	*He answers emails in a timely fashion*	answers emails in a timely fashion
21	*He answers emails in a timely fashion*	*responding to email promptly (even when you don't want to)
22	*He answers emails in a timely fashion*	answers e-mails right away
23	*He answers emails in a timely fashion*	*available
24	*He answers emails in a timely fashion*	answer emails fast
25	*He believes in the open-source ideology*	*wants to be part of the open-source community
26	*He believes in the open-source ideology*	not to idealogical
27	*He builds personal relations with contributors*	Empaty (getting to know people on the project on a personnal level)}
28	*He builds personal relations with contributors*	*the ability to work |with others remotely and still maintain and sense of |connectiveness
29	*He cares about settling conflicts between developers*	settles disagreements between other developers
30	*He cares about settling conflicts between developers*	Mediator
31	*He cares about settling conflicts between developers*	Negotiator
32	*He cares about settling conflicts between developers*	Mediates in conflicts
33	*He cares about settling conflicts between developers*	Mediating conflicts
34	*He cares about settling conflicts between developers*	solve disputes
35	*He cares about settling conflicts between developers*	right arbitrator
36	*He cares about settling conflicts between developers*	arbitrer (for example betwenn members)
37	*He considers aestheticall issues*	considers aestheticall issues
38	*He coordinates the efforts of the contributors (e.g to avoid overlapping contribution and to make things work togheter)*	to coordinate things

39	*He coordinates the efforts of the contributors (e.g to avoid overlapping contribution and to make things work togheter)*	coordinate single project groups
40	*He coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work togheter)*	coordinate releases
41	*He coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work togheter)*	coordinate activites: many contributors on a project mean there are many things going on at once. The project leader coordinates the contributions~ ensuring each contribution works together
42	*He coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work togheter)*	so is avoiding duplicate work among developers
43	*He coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work togheter)*	make sure different people working on the same issue talk to each other
44	*He coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work togheter)*	coordinates releases
45	*He decides what code/features to include or not to include*	Retire features/code/functionality when it is no longer used
46	*He decides what code/features to include or not to include*	decides which patches to apply
47	*He decides what code/features to include or not to include*	decide what to include and not include
48	*He decides what code/features to include or not to include*	makes decisions on a day-to-day basis about what code to include/not include
49	*He decides what code/features to include or not to include*	decide features to include
50	*He discourages heated arguments*	discourages heated arguments
51	*He discourages heated arguments*	looking forward(don't hang in old debats)
52	*He discusses in the open various approaches with other coders*	discuss with others on how to resolve bugs
53	*He discusses in the open various approaches with other coders*	discusses various approaches with other coders
54	*He discusses in the open various approaches with other coders*	fosters open discussion and debate
55	*He discusses in the open various approaches with other coders*	communicates his ideas|invites others to share their opinion on technical issues
56	*He discusses in the open various approaches with other coders*	discuss with others on techinical matters
57	*He divides the work among contributors*	divide the work
58	*He does not rule out people when poeple don't fit his vision*	Not ruling out other people's projects even when they don't fit with his vision or immediate goals
59	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	keeps deadlines otherwise nothing is gonna work
60	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	make contributors observe "features freezes"
61	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	*Time management. As an open source leader you have to deal with a very large number of volunteers~ each of which can only contribute a small ammount of time. Some times I feel like I am more a manager than a developer.
62	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	time-management - it is important to keep deadlines
63	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	keeps deadlines
64	*He emphasizes the meeting of deadlines (e.g. "feature freezes")*	sets terms to be respected (see the GNOME time-based releases)
65	*He encourages participation to the project*	Encourage new developers
66	*He encourages participation to the project*	encourages others to participate
67	*He encourages participation to the project*	encourages participation

68	*He encourages participation to the project*	Ability to engage users and people with all skill levels and time constraints. There must be a place for everyone.
69	*He encourages participation to the project*	Encouraging people to contribute
70	*He encourages participation to the project*	encourage people to submit patches
71	*He establishes partnership with companies and others projects*	establishing partnership with companies and other projects
72	*He establishes the roadmap/plan for the development*	Establishes the roadmap/plan for the development
73	*He establishes the roadmap/plan for the development*	Defining project roadmaps is important
74	*He establishes the roadmap/plan for the development*	updating roadmaps
75	*He establishes the roadmap/plan for the development*	sets milestones
76	*He establishes the roadmap/plan for the development*	Work on product roadmap
77	*He establishes the roadmap/plan for the development*	A clear road-map of what has to be done~ and what is most important. For example~ in the project I'm running I have a list of things "to do".
78	*He establishes the roadmap/plan for the development*	organizes a project's goals
79	*He establishes the roadmap/plan for the development*	Publicizes the goals/mission/roadmap/plan so that teammembers are clear on what is important to work on
80	*He explains publicly his choices*	explain choices which were taken or no
81	*He favors information exchanges*	favors information exchange
82	*He gives feedbacks on the works of others*	gives balanced feedback to active developers
83	*He has a general overview of the project*	has overview
84	*He has a general overview of the project*	Ability to see the forest for the trees
85	*He has a general overview of the project*	A global vision
86	*He has a mature sense of responsibilities*	Mature
87	*He has a mature sense of responsibilities*	Maturity
88	*He has a mature sense of responsibilities*	mature sense of responsisbilities
89	*He has a mature sense of responsibilities*	*responsible
90	*He has a mature sense of responsibilities*	*accountable
91	*He has a vision, a future perspective for the project*	Visionary
92	*He has a vision, a future perspective for the project*	good overview of project goals
93	*He has a vision, a future perspective for the project*	Establishes the mission for the project
94	*He has a vision, a future perspective for the project*	a sense of perspective
95	*He has a vision, a future perspective for the project*	clearly recognizes the scope and tasks of the project
96	*He has a vision, a future perspective for the project*	good vision
97	*He has a vision, a future perspective for the project*	intuitive feel for the direction and long term goals of the project
98	*He has a vision, a future perspective for the project*	Vision
99	*He has a vision, a future perspective for the project*	focused on certain pre-established goals
100	*He has a vision, a future perspective for the project*	clear vision of the project and it's future
101	*He has a vision, a future perspective for the project*	good vision
102	*He has a vision, a future perspective for the project*	has a vision
103	*He has a vision, a future perspective for the project / He decides which features to include and not to include*	*decide about future changes
104	*He has an independent character*	independent character
105	*He has been under a good project leader for a long time*	has been under a good project leader himself for a long time
106	*He has cultural sensitivity*	Basic cultural sensitivity (if the project is global)
107	*He has cultural sensitivity*	Multicultural
108	*He has the ability to think like a user (to approximate their needs)*	*the ability to think like a user
109	*He helps contributors resolve technical problems*	Helpful
110	*He helps contributors resolve technical problems*	elucidates doubts from other developers

111	*He helps contributors resolve technical problems*	helps developers of his software
112	*He helps contributors resolve technical problems*	Helpful
113	*He helps contributors resolve technical problems*	help developers with specific problems
114	*He helps contributors resolve technical problems*	willing to help
115	*He helps contributors resolve technical problems*	available for questions by developers / users
116	*He helps contributors resolve technical problems*	help with answering questions
117	*He helps contributors resolve technical problems*	support the project/product (I mean some kind of hotline)
118	*He helps contributors resolve technical problems*	helps contributors
119	*He helps contributors resolve technical problems*	help with fixing bugs
120	*He is a perfectionist*	Perfectionist
121	*He is active in the development of code*	*lead-by-doing
122	*He is active in the development of code*	Coding
123	*He is active in the development of code*	is also a developer
124	*He is active in the development of code*	participates to the software development
125	*He is active in the development of code*	write lots of good code
126	*He is active in the development of code*	develops and develops
127	*He is active in the development of code*	active in coding
128	*He is active in the development of code*	writes code (usually the major part!)
129	*He is active in the development of code / He is dedicated to the project (time, effort […])*	active in coding and dedicated
130	*He is always looking for improvements*	aways looking for improvements
131	*He is an experienced coder*	Experience in coding
132	*He is an experienced coder*	Experienced
133	*He is an experienced coder*	experience with software design
134	*He is an experienced coder*	Experienced
135	*He is an experienced coder*	Experiences
136	*He is an experienced coder*	experienced coder
137	*He is calm and equilibrate*	Basic mental stability (ie: no behavioural disorders)
138	*He is calm and equilibrate*	Calm
139	*He is calm and equilibrate*	never nervous
140	*He is collaborative*	able to work with other people (a lot of developers lack this skill). Definitely not a "prima donna" kind of developer.
141	*He is collaborative*	wants to collaborate
142	*He is committed to a quality product (he makes sure bugs are corrected)*	constantly reviews code for quality testability
143	*He is committed to a quality product (he makes sure bugs are corrected)*	Fixing bugs
144	*He is committed to a quality product (he makes sure bugs are corrected)*	considers technical issues
145	*He is committed to a quality product (he makes sure bugs are corrected)*	track bugs
146	*He is committed to a quality product (he makes sure bugs are corrected)*	committed to a quality product
147	*He is committed to a quality product (he makes sure bugs are corrected)*	Testing
148	*He is committed to a quality product (he makes sure bugs are corrected)*	tackles the reported errors
149	*He is committed to a quality product (he makes sure bugs are corrected)*	review the quality of the code and patches
150	*He is committed to a quality product (he makes sure bugs are corrected)*	making sure bugs are effectively eliminated
151	*He is committed to a quality product (he makes sure bugs are corrected)*	bug-busting-business
152	*He is committed to a quality product (he makes sure bugs are corrected)*	Debug
153	*He is committed to a quality product (he makes sure bugs are corrected)*	attentive to bug reports from users

154	*He is committed to a quality product (he makes sure bugs are corrected)*	work on open bugs
155	*He is concerned with getting the things done*	*moves the project forward
156	*He is concerned with getting the things done*	Ability to focus until job is completly done
157	*He is concerned with getting the things done*	Deliver
158	*He is concerned with getting the things done*	wants things get done
159	*He is dedicated to the project (time, effort [..])*	dedicated (time/effort)
160	*He is dedicated to the project (time, effort [..])*	Stays in touch with or at least aware of the major activities in the project on a daily basis
161	*He is dedicated to the project (time, effort [..])*	dedicated to project & has plenty of time to commit to it.
162	*He is dedicated to the project (time, effort [..])*	Involvement
163	*He is dedicated to the project (time, effort [..])*	maintains the project actively
164	*He is dedicated to the project (time, effort [..])*	donate money and time to the project
165	*He is dedicated to the project (time, effort [..])*	has enough time to spend on it
166	*He is dedicated to the project (time, effort [..])*	Creates initial code to prove that he/she is willing to spend time/work on it
167	*He is dedicated to the project (time, effort [..])*	*a sense of ownership of the project
168	*He is dedicated to the project (time, effort [..])*	Dedication
169	*He is dedicated to the project (time, effort [..])*	very dedicated in terms of time
170	*He is dedicated to the project (time, effort [..])*	put a lot of time on the project
171	*He is dedicated to the project (time, effort [..])*	dedicated to the project
172	*He is delegative - he doesn't try to do all the work on his own*	is able to delegate responsibility (trusts others)
173	*He is delegative - he doesn't try to do all the work on his own*	Ability to delegate work. You can't completley run the whole project. Letting people make decisions on their own is important to keep them interested~ give them a sense of ownership (see the section on volunteer pride) and ease of your schedule.}
174	*He is delegative - he doesn't try to do all the work on his own*	Ability to delegate tasks to people with less experience
175	*He is delegative - he doesn't try to do all the work on his own*	Delegating tasks
176	*He is delegative - he doesn't try to do all the work on his own*	Being able to delegate tasks on others (not try to do all the work on his own)}
177	*He is delegative - he doesn't try to do all the work on his own*	delegate to other people
178	*He is delegative - he doesn't try to do all the work on his own*	willing to delegate
179	*He is delegative - he doesn't try to do all the work on his own*	the ability to track and delegate responsibilities to others
180	*He is delegative - he doesn't try to do all the work on his own*	willing to delegate control/authority
181	*He is delegative - he doesn't try to do all the work on his own*	delegates authority to other project members
182	*He is delegative - he doesn't try to do all the work on his own*	to delegate
183	*He is diplomatic*	Diplomatic
184	*He is diplomatic*	Diplomacy
185	*He is easy to understand*	good with people over email
186	*He is easy to understand*	good communicator, easy to understand
187	*He is easy to understand*	writes well
188	*He is easy to understand*	good communicator
189	*He is easy to understand*	speaks clear
190	*He is easy to understand*	communicates well
191	*He is easy to understand*	makes himself understood by the others
192	*He is easy to understand*	able to communicate in a clear manner
193	*He is easy to understand*	Excellent communication skills
194	*He is easy to understand*	writes in a way everyone understands
195	*He is enthusiast for the project and excitable*	is proud of the project

196	*He is enthusiast for the project and excitable*	Excitable
197	*He is enthusiast of the project and excitable*	enthusiasm for the project
198	*He is friendly and approachable*	Very reachable for the project members
199	*He is friendly and approachable*	Nice
200	*He is friendly and approachable*	Friendly
201	*He is friendly and approachable*	Conversant
202	*He is friendly and approachable*	a good friend in the project to talk to
203	*He is friendly and approachable*	socially proficient
204	*He is friendly and approachable*	good social sense
205	*He is friendly and approachable*	Friendly kind of guy who can talk with anybody
206	*He is friendly and approachable*	Friendly
207	*He is humble*	Humble
208	*He is humble / He accepts external contributions (he is willing to say "that piece of work is better than mine")*	humility. In particular the ability to say "that pice of work is better than mine"
209	*He is humorous - development is fun with him*	A sense of humour. Development must be fun. This is important for any volunteer-based project.
210	*He is humorous - development is fun with him*	sense of humour
211	*He is intelligent*	Intelligent
212	*He is intelligent*	Intelligent
213	*He is intelligent*	Intelligent
214	*He is knowledgable on legal aspects*	understands legal implications of IP
215	*He is knowledgable on technical aspects of the project*	has expertise of technical project details
216	*He is knowledgable on technical aspects of the project*	in depth knowledge of the project
217	*He is knowledgable on technical aspects of the project*	understanding of project requisites
218	*He is knowledgable on technical aspects of the project*	knows the structure of the implementation
219	*He is knowledgable on technical aspects of the project*	must know a lot about the topic
220	*He is knowledgable on technical aspects of the project*	knows every technical aspect of the project
221	*He is knowledgable on technical aspects of the project*	good technical knowledge in some domain
222	*He is knowledgable on technical aspects of the project*	knowledge of protocols and standards
223	*He is not easily drawnable into flamefests*	thick skin i.e. not easily drawn into flamefests
224	*He is not easily drawnable into flamefests*	thick skin
225	*He is optimist*	optimism
226	*He is optimist*	riesce a trasmettere fiducia nel progetto
227	*He is organised*	Organized
228	*He is organized*	Organized
229	*He is patient when dealing with contributors*	patient when explaining
230	*He is patient when dealing with contributors*	Patience
231	*He is patient when dealing with contributors*	patient when dealing with contributors
232	*He is patient when dealing with contributors*	Patience
233	*He is patient when dealing with contributors*	Tolerance
234	*He is patient when dealing with contributors*	shows a lot a patience with developers
235	*He is patient when dealing with contributors*	patience
236	*He is patient when dealing with contributors*	patient working with codevelopers
237	*He is patient when dealing with contributors*	*the ability |to deal with people without hating them
238	*He is patient when dealing with contributors*	Patient
239	*He is polite and respectful*	Keeping polite during conflicts
240	*He is polite and respectful*	polite and respectful
241	*He is polite and respectful*	respect for other's opinions
242	*He is polite and respectful*	Respectful
243	*He is polite and respectful / He is concerned with getting the things done / He discourages heated arguments*	*diplomacy
244	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	present in the mailing list
245	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	Day and night monitoring of project chat room.

246	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	Availability (constant presence~ at least perceived)
247	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	online regularly
248	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	frequently converse (live?) on IRC
249	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	being a visible leader on mailing lists and other forms of communications
250	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	makes him/herself available on project mailing lists or discussion channels
251	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	communicate frequently via email/IRC/IM
252	*He is present in the places where development occurs (mailing-lists/IRC/etc.)*	visible in the mailing list
253	*He is prone to step out when needed*	Takes a vacation when necessary
254	*He is realistic on what could be done and achieved / He sets out the overall direction of the project*	set direction / realistic when setting goals
255	*He is self-motivated*	self-motivated
256	*He is technically accurate*	Accurate
257	*He is technically accurate*	Thorough
258	*He is technically accurate*	Rigorous
259	*He is technically competent - he writes real good code*	coding skills
260	*He is technically competent - he writes real good code*	technically competent
261	*He is technically competent - he writes real good code*	technically good (but not necessary an expert)
262	*He is technically competent - he writes real good code*	Skilled
263	*He is technically competent - he writes real good code*	Competente
264	*He is technically competent - he writes real good code*	Technical expertise
265	*He is technically competent - he writes real good code*	good programmer
266	*He is technically competent - he writes real good code*	professional level contributions: low-quality contributions leaves other contributors doubting the leader's judgement and credibility
267	*He is technically competent - he writes real good code*	good technical judgement
268	*He is technically competent - he writes real good code*	able to write real code
269	*He is technically competent - he writes real good code*	produces good source code
270	*He is technically competent - he writes real good code*	competent coder
271	*He is technically competent - he writes real good code*	good engineering skills
272	*He is technically competent - he writes real good code*	Competent
273	*He is technically competent - he writes real good code*	competent programmer
274	*He is technically competent - he writes real good code*	not a perfect programmer
275	*He is technically competent - he writes real good code*	good architect
276	*He is technically competent - he writes real good code*	good coder
277	*He is technically competent - he writes real good code*	technical abilities to gain respect|of project participants
278	*He is technically competent - he writes real good code*	technically proficient
279	*He is technically competent - he writes real good code*	writes quality code
280	*He is technically competent - he writes real good code*	Knowledge
281	*He is technically competent - he writes real good code*	Knowledgeable
282	*He is technically competent - he writes real good code*	good technical design skills
283	*He is trustworthy*	Trustable
284	*He is trustworthy*	Trustworthy
285	*He is willing to mentor others*	helps out new developers find their way
286	*He is willing to mentor others*	educate new developers
287	*He is willing to mentor others*	willing to mentor others
288	*He is willing to mentor others*	good teacher
289	*He is willing to mentor others*	willing to educate newcomers
290	*He is willing to mentor others*	teaches to new contributors
291	*He is willing to mentor others / He sets out the overall direction of the project*	*provides leadership to less experienced developers
292	*He is willing to prune to improve growth*	Willingness to prune to improve growth

293	*He keeps everyone informed on the project directions and achievements*	keeps users informed about the project progress / plan
294	*He keeps everyone informed on the project directions and achievements*	keep developers informed about the project direction
295	*He keeps everyone informed on the project directions and achivements*	keep everybody informed
296	*He keeps everyone informed on the project directions and achivements*	providing an overview of the project goals and achievements
297	*He keeps everyone informed on the project directions and achivements*	keep track of where project is and where it's going
298	*He keeps everyone informed on the project directions and achivements*	keep users informed about progress / plans
299	*He keeps everyone informed on the project directions and achivements*	keeping people informed of the project plans and directions
300	*He keeps the project interesting and alive*	Keeps project interesting and alive
301	*He knows about cutting edge techniques*	Cutting edge knowlege of new techinqes
302	*He knows about cutting edge techniques*	stay in front on new tehc.
303	*He knows at least one second language*	Know at least one second language (makes one appreciate how english may be for other people)
304	*He knows at least one second language*	knows different languages
305	*He listens to what contributors have to say*	listen to maintainers
306	*He listens to what contributors have to say*	listens to people
307	*He listens to what contributors have to say*	good listener
308	*He listens to what contributors have to say*	listens to user demands
309	*He listens to what contributors have to say*	don't listen to talkers
310	*He listens to what contributors have to say*	listen to contributors
311	*He listens to what contributors have to say*	_listen_ to his codevelopers
312	*He makes decisions - public debate could drag ineffectively for weeks*	Ability to say "the buck stops here~ and it stops now"
313	*He makes decisions - public debate could drag ineffectively for weeks*	decide (to keep the project going forward)
314	*He makes decisions - public debate could drag ineffectively for weeks*	make decisions: somebody has to decide when design debates have reached a conclusion and decide the course to take
315	*He makes decisions - public debate could drag ineffectively for weeks*	*the ability to make decisions
316	*He makes decisions - public debate could drag ineffectively for weeks*	*decisive
317	*He makes decisions - public debate could drag ineffectively for weeks*	takes decisions, someone must to do that
318	*He makes decisions - public debate could drag ineffectively for weeks*	must be able to|make decissions - public debate can drag for WEEKS
319	*He makes no discriminations*	non-discriminating
320	*He makes sure that documentation of the software is available*	Encourages supporting activities lige FAQ maintenance~ HOWTO writing etc. as much as coding
321	*He makes sure that documentation of the software is available*	constantly reviews code for documentation
322	*He makes sure that documentation of the software is available*	ensures project documentation is available and of reasonable quality
323	*He makes sure that documentation of the software is available*	documents the project in its current status
324	*He makes sure that guidelines and coding procedures are available to developers*	sets coding guidlines for contributors
325	*He makes sure that guidelines and coding procedures are available to developers*	provide potential contributors with detailed intructions on how to write patches
326	*He makes sure that guidelines and coding procedures are available to developers*	set up guidelines for developers
327	*He makes sure that guidelines and coding procedures are available to developers*	is clear about the guidelines and the coding procedures
328	*He makes sure that guidelines and coding procedures are available to developers*	work on standards to be used by developers

329	*He motivates contributors*	can motivate people
330	*He motivates contributors*	motivates people
331	*He motivates contributors*	good motivation skills
332	*He motivates contributors*	motivating/Inspiring partners
333	*He offers new approaches to problems*	points out possible corrections
334	*He offers new approaches to problems*	suggest how to resolve problems
335	*He offers new approaches to problems*	generate ideas: if there is a problem and nobody has an idea about how to fix it~ it will often be the leader who steps up to lead the discussion to find a solution
336	*He offers new approaches to problems*	suggests corrections
337	*He offers new approaches to problems*	always has an idea on how fix a problem
338	*He offers new approaches to problems*	tells how to fix nasty bugs (nobody has been able to fix)
339	*He offers new approaches to problems*	offers solutions
340	*He organize meetings (IRC, email [..])*	organizing meetings (IRC mail etc...)
341	*He organizes the project activities*	good organizational skills
342	*He organizes the project activities*	organizes well contributors tasks
343	*He organizes the project activities*	good organization skills
344	*He organizes the project activities*	organizational skills
345	*He persuades others that the choosen idea is best*	persuades other the choosen idea is best
346	*He praises developers for their contribution*	regularly praises the developers
347	*He praises developers for their contribution*	Retires project members with full honours when they drop away
348	*He praises developers for their contribution*	thanks contributors for their contributions
349	*He praises developers for their contribution*	Help the volunteers get a sense of pride and accomplishments. Praising their efforts is good~ but here are other forms.
350	*He praises developers for their contribution*	and never belittle other people's work most free software is done in spare time by volunteers
351	*He praises developers for their contribution*	knows to thank people
352	*He promotes the project (advertises it, maintains PR etc)*	understands|marketing
353	*He promotes the project (advertises it, maintains PR etc)*	advertise the project
354	*He promotes the project (advertises it, maintains PR etc)*	promotes the project
355	*He promotes the project (advertises it, maintains PR etc)*	doing marketing
356	*He recruits new project members*	Recruits new developers
357	*He recruits new project members*	find good people that will help the project and recruit them
358	*He recruits new project members*	recruits additional personnel to the project
359	*He recruits new project members*	seeks new talents
360	*He releases frequently*	organizes releases of codebase frequently
361	*He represents the project in conferences and the like*	present at conferences
362	*He represents the project in conferences and the like*	represent the project in the events
363	*He represents the project in conferences and the like*	represent the project
364	*He represents the project in conferences and the like*	Representation
365	*He represents the project in conferences and the like*	present for the project in the major happenings
366	*He reviews the work of others*	review patches
367	*He reviews the work of others*	is a good reviewer
368	*He reviews the work of others*	reviewing patches and code check-ins
369	*He reviews the work of others / He gives feedbacks on the work of others*	check on the work of others and give them feedback
370	*He sets out the overall direction of the project*	sets the direction of the project
371	*He sets out the overall direction of the project*	points out the project's real goal every time someone gets lost
372	*He sets out the overall direction of the project*	Sets out the overall direction
373	*He sets out the overall direction of the project*	guides the others

374	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	ensures website is kept up to date
375	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	creates and maintains a good mailing list
376	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	creates and maintains a clear, concise website
377	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	organize the project (paperwork^ bank account^ web site^ cvs repository^)
378	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	creates infrastructure
379	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	Creates a community somehow (website~ forums etc.)
380	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	sets up / provides an infrastructure (cvs etc.)
381	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	Maintain source control (CVS-branches)
382	*He sets up / maintains an infrastructure (cvs/website/mail-list)*	work on infrastructure to be used by developers
383	*He starts a project out of a desire to learn or expand on knowledge of a new technology*	starts a project out of a desire to learn or expand on knowledge of a new technology
384	*He starts a project out of a desire to learn or expand on knowledge of a new technology*	wants to learn
385	*He takes care of building the final package*	packaging skills
386	*He takes care of building the final package*	packaging releases
387	*He throws project members out if they have negative overall contribution*	Throw project members out if they have negative overall contribution
388	*He throws project members out if they have negative overall contribution*	and isolating those you can't work together yet are valuable to the project
389	*His mind is open to new solutions*	open-minded
390	*His mind is open to new solutions*	does not discard any alternative solution (openminded)
391	*His mind is open to new solutions*	open minded
392	*His mind is open to new solutions*	open for ideas

Appendix E – Rating-based questionnaire

Page 1 of 4

Leaders' behavior in the free software/open-source community

Thank you for participating in this survey! The results of the following questionnaire will be used for my **master thesis** on leadership in the free software/open-source environment.

This questionnaire will require around 10 minutes to complete. Note that NEITHER your contribution NOR the described project leader will be identified IN ANY WAY.

If you are interested - you will have the chance to read a brief description of the objective of my project after having submitted the questionnaire.

There is a plain text version of the questionnaire available here. However, I would warmly recommend you to continue with the following web version of it ;)

Thank you for your time!

Gianluca Bosco g.bosco@inwind.it
Technical University of Denmark (DTU)
Department of Manufacturing Engineering and Management

1) What's your age?

2) What's your nationality?

3) How many project have you been contributing to, as a free software/open-source developer? (e.g. 6)

4) For how long have you been contributing to free software/open-source projects? (e.g. 3 months; 10 years;)

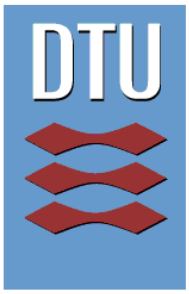
Thesis and questionnaire author: Gianluca Bosco
Thesis supervisor: Ph.D. Kasper Edwards
Technical University of Denmark (DTU) - Department of Manufacturing Engineering and Management

Next Page

Click Here to Conduct Your Own Survey

Page 2 of 4

Leaders' behavior in the free software/open-source community



5) Regarding the project in which you are most active, how satisfied are you with the project leader? (If you yourself are the project leader please think of a project where you do not or didn't have a leading position.)

- I'm NOT satisfied at ALL
- I'm unsatisfied
- I'm sufficiently satisfied
- I'm satified
- I'm VERY satisfied

6) Please, rate the following behaviors and traits depending on *how well they describe* the project leader you evaluated in the previous question:

	It doesn't describe my project leader at all 1	- 2	- 3	- 4	- 5	- 6	- 7	- 8	It describes my project leader very well 9
He or she discusses in the open various approaches with other coders	○	○	○	○	○	○	○	○	○
He or she accepts external contributions (he or she is willing to say "that piece of work is better than mine")	○	○	○	○	○	○	○	○	○
He or she is committed to a quality product (he or she makes sure bugs are corrected)	○	○	○	○	○	○	○	○	○
He or she makes sure that guidelines and coding procedures are available to developers	○	○	○	○	○	○	○	○	○
He or she is patient when dealing with contributors	○	○	○	○	○	○	○	○	○
He or she answers emails in a timely fashion	○	○	○	○	○	○	○	○	○
He or she is easy to understand	○	○	○	○	○	○	○	○	○
He or she is delegative - he or she doesn't try to do all the work on his own	○	○	○	○	○	○	○	○	○
He or she sets out the overall direction of the project	○	○	○	○	○	○	○	○	○
He or she encourages participation in the project	○	○	○	○	○	○	○	○	○
He or she is knowledgable on technical aspects of the project	○	○	○	○	○	○	○	○	○
He or she cares about settling conflicts between developers	○	○	○	○	○	○	○	○	○

Thesis and questionnaire author: Gianluca Bosco
Thesis supervisor: Ph.D. Kasper Edwards
Technical University of Denmark - Department of Manufacturing Engineering and Management

Next Page

Click Here to Conduct Your Own Survey

Page 3 of 4

Leaders' behavior in the free software/open-source community

7) 2/3, continues from previous page

	It doesn't describe my project leader at all 1	- 2	- 3	- 4	- 5	- 6	- 7	- 8	It describes my project leader very well 9
He or she is friendly and approachable	○	○	○	○	○	○	○	○	○
He or she is technically competent - he or she writes real good code	○	○	○	○	○	○	○	○	○
He or she keeps everyone informed on the project directions and achievements	○	○	○	○	○	○	○	○	○
He or she offers new approaches to problems	○	○	○	○	○	○	○	○	○
He or she coordinates the efforts of the contributors (e.g to avoid overlapping contributions and to make things work together)	○	○	○	○	○	○	○	○	○
He or she helps contributors resolve technical problems	○	○	○	○	○	○	○	○	○
He or she is an experienced coder	○	○	○	○	○	○	○	○	○
He or she decides what code/features to include or not to include	○	○	○	○	○	○	○	○	○
He or she listens to what contributors have to say	○	○	○	○	○	○	○	○	○
He or she accepts comments and suggestions	○	○	○	○	○	○	○	○	○
He or she acts as the spokesperson of the project	○	○	○	○	○	○	○	○	○
He or she is dedicated to the project (time, effort, etc.)	○	○	○	○	○	○	○	○	○

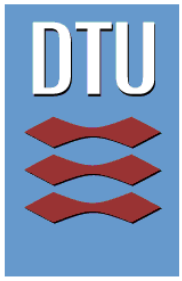
Thesis and questionnaire author: Gianluca Bosco
Thesis supervisor: Ph.D. Kasper Edwards
Technical University of Denmark (DTU) - Department of Manufacturing Engineering and Management

Next Page

Click Here to Conduct Your Own Survey

Page 4 of 4

Leaders' behavior in the free software/open-source community



8) 3/3, continues from previous page

	It doesn't describe my project leader at all 1	- 2	- 3	- 4	- 5	- 6	- 7	- 8	It describes my project leader very well 9
He or she is willing to mentor others	○	○	○	○	○	○	○	○	○
He or she is concerned with getting things done	○	○	○	○	○	○	○	○	○
He or she emphasizes the meeting of deadlines (e.g. "feature freezes")	○	○	○	○	○	○	○	○	○
He or she praises developers for their contributions	○	○	○	○	○	○	○	○	○
He or she makes decisions - public debate could drag ineffectively for weeks	○	○	○	○	○	○	○	○	○
He or she is present in the places where development occurs (mailing-lists/IRC/etc.)	○	○	○	○	○	○	○	○	○
He or she establishes the roadmap/plan for development	○	○	○	○	○	○	○	○	○
He or she sets up and/or maintains an infrastructure (cvs/website/mailing-list)	○	○	○	○	○	○	○	○	○
He or she is active in the development of code	○	○	○	○	○	○	○	○	○
He or she is polite and respectful	○	○	○	○	○	○	○	○	○
He or she has a vision, a future perspective for the project	○	○	○	○	○	○	○	○	○
He or she represents the project in conferences and the like	○	○	○	○	○	○	○	○	○

9) If you like, you can use the following field to post comments concerning this questionnaire. For example things you did not like, vague questions, etc

10) If you are interested receiving a link to a copy of the final report, please insert your email address:

Thesis and questionnaire author: Gianluca Bosco
Thesis supervisor: Ph.D. Kasper Edwards
Technical University of Denmark (DTU) - Department of Manufacturing Engineering and Management

Submit

Click Here to Conduct Your Own Survey

Your answers have been recorded. Now you can close the windows or, if you are interested, read the description of my project.

The objective of my thesis is to identify any or more prototypes of "good open-source project leader" as it resides in the minds of the contributors, in terms of personality traits, personal attributes and behaviours. I have choosen a social-cognitive approach to identify the personal believes implicitly utilized by contributors to classify open-source project leaders as "good" ones. The main feature of such approach is its focus on the *contributor's perceptual processes*.

Generic observers have been shown to *classify persons* based on their matching to prototypes, made up by personality traits and behaviours. When a match to a prototype occurs, the person is classified accordingly. For example, we all have an idea of what an "extravert" is in terms of personality traits and behaviours. When we meet someone that is very talkative and smiling, we classify him as an extravert, in a more or less conscious way.

Being classified into a category, has consequences in the way persons interact with each other. Prototypes allow observers to develop *consistent behavioural expectations* towards categorized persons. In the same way, subsequent behaviour of the categorized person is *interpreted* accordingly to the relevant prototype. It is likely that a person classified as an "extravert" will be invited to the next party on saturday. Mistakes of an "effective political leader" are more likely to be interpreted as due to uncontrollable causes than to his personal responsability, by the electors.

Prototypes are built through experience and they are the result of *different environmental factors* like national and organizational culture, task characteristics, level of group identification and self-views.

Going back to the open-source environment: what are the relevant behaviors and traits characterizing a good project leader? Is there only one shared prototype of good open-source project leader, or are there many different ones? What's the importance of each behavior? Can the contributor's national cultural belonging and level of experience in contributing to open-source projects explain differences in prototypes?

If you have any kind of feedback or if you are interested in receiving a copy of the final report - just drop me an email to this address g.bosco@inwind.it. I will be pleased to answer you back ;)

Gianluca Bosco

Appendix F – Open-source projects invited to the rating-based questionnaire.

A public email of invitation to the second questionnaire set up for this work has been posted in the major development mailing lists or newsgroups dedicate to the following projects (in alphabetic order):

Alzabo
Bugzilla
Compiere
Emacs
Epiphany
Fvwm
Gimp
Gnucash
Horde
Jabber
Kaffe
KDE
Kopete
Kwin
Mailman
Mono
MySQL
NetBSD
Openoffice
Orbit
Perl
PHP
Plone
Prelude
Spamassassin
Squid
Tcl
Webmin
XFree86
Zope

Appendix G – Unrotated Factor Matrix

Unrotated Factor Matrix

	Factor		
	1	2	3
discusses_openly	.619	.055	-.331
accepts_contributions	.551	.048	-.411
patient	.726	-.175	-.275
answers_emails_promptly	.657	-.295	.003
easy_to_understand	.662	-.316	-.176
delegative	.534	.003	-.273
settle_conflicts	.624	-.280	-.041
friendly	.580	-.208	-.507
keeps_informed	.679	-.052	.117
listens	.790	-.266	-.208
accepts_suggestions	.718	-.115	-.252
praises_developers	.650	-.152	.088
polite	.649	-.138	-.322
represents_the_project	.363	-.144	.371
talks_for_the_group	.411	-.256	.262
knowleadgeable	.420	.563	-.214
experienced	.179	.826	-.237
competent	.322	.634	-.324
sets_direction	.510	.248	.326
has_vision	.653	.017	.252
mentor_others	.548	-.169	.234
helps_contributors	.635	.216	-.024
offers_new_approaches	.615	.217	.000
decides_what_to_include	.145	.528	.273
makes_decisions	.430	.085	.564
maintains_infrastructure	.362	.071	.254
establishes_roadmap	.563	.212	.450
coordinates_work	.615	-.069	-.030
makes_guidelines	.428	-.064	.242
commited_to_quality	.448	.323	.121
encourgaes_participation	.766	.059	-.038
dedicated	.292	.106	.369
get_things_done	.612	.000	.351
emphasizes_deadlines	.461	-.078	.248
present	.267	.079	.102
active	.260	.539	-.084

Table 7 - Unrotated factor matrix of the three extracted factors

References

Barsalou, L.W. (1983). Ad hoc categories. *Memory and Cognition, 11, 211-227.*

Bartol, K.M., & Butterfield, D.A. (1976). Sex effects in evaluating leaders. *Journal of Applied Psychology, 61(4), 446-454.*

Bass, B.M., (1990). *Bass & Stogdill's handbook of leadership: theory, research, and managerial applications (3rd ed.).* New York: Free Press.

Bowers, D.G. and Seashore, S.E. (1966). Predicting organizational effectiveness with a four factor theory of leadership. *Administrative Science Quarterly, 11, 238-263.*

Calder, B. J. (1977). An attribution theory of leadership. In B. M. Staw & G. R. Salancik (Eds.), *New directions in organizational behavior.* Chicago: St. Clair.

Cantor, N., & Mischel, W. (1979). Prototypes in person perception. In L. Berkowitz (Eds.), *Advances in experimental psychology* (vol. 12). New York: Academic Press.

Christiansen, K.R., 2003, Interview conducted the 8th of October.

Cohen, C.E. (1983). Inferring the characteristics of other people: categories and attribute accessibility. *Journal of Personality and Social Psychology, 44, 34-44.*

Cronshaw, S.F., & Lord, R.G. (1987). Effects of categorization, attribution, and encoding processes on leadership perceptions. *Journal of Applied Psychology, 71(1), 97-106.*

Deci, E.L., & Ryan, R.M. (1985). *Intrinsic motivation and self-determination in human behavior.* New York: Plenum Press.

Eden, D., & Leviatan, U. (1975). Implicit leadership theory as a determinant of the factor structure underlying supervisory behavior scales. *Journal of Applied Psychology, 60, 736-741.*

Edwards, K., (2000). When Beggars Become Choosers. Article available at: http://firstmonday.org/issues/issue5_10/edwards/index.html

Edwards, K., (2001). Towards a Theory for Understanding the Open Source Software Phenomenon. Paper available for download at: http://edwards.dk/towards.pdf

Edwards, K., (2003). Technological Innovation in Software Industry – Open Source Software. Ph.D. thesis available for download at: http://edwards.dk/thesis.pdf

Fielding, R.T. (1999). Shared leadership in the Apache project. *Communications of the ACM, 42(4), 42-43.*

Gerstner, C.R., & Day, D.V. (1994). Cross-cultural comparison of leadership prototypes. *Leadership Quarterly, 5(2), 121-134.*

Gibb, C.A. (1954). Leadership. In G. Lindzey (ed.), *Handbook of social psychology (vol. 2),* Reading: Addison-Wesley.

Hall, R.J., Workman, J.W., & Marchioro, C.A. (1998). Sex, task, and behavioral flexibility effects on leadership perceptions. *Organizational Behavior and Human Decision Processes, 74, 1-32.*

Hollander. E.P., & Julian, J.W. (1969). Contemporary trends in the analysis of leadership processes. *Psychological Bulletin, 71, 387-397.*

Holmström, B. (1999). Managerial incentive problems: a dynamic perspective. *Review of Economic Studies, 66, 169-182.*

Kamp, P., 2003, Interview conducted the 10th of October.

Kamp, P., 2004, Email dated Monday, 23 February 2004 13:13:00.

Katz, D., & Kahn, R.L. (1978). *The social psychology of organizations (2nd ed.).* New York: John Wiley.

Kelly, G.A. (1963). *A theory of personality: the psychology of personal constructs.* New York: Norton.

Lakhani, K.R., & Wolf, R.B. (2003). Why hackers do what they do: understanding motivation efforts in Free/Open Source software projects. MIT Sloan Working Paper No. 4425-03 available for download at http://ssrn.com/abstract=443040

Lakoff, G. (1972). Hedges: A study in meaning criteria and the logic of fuzzy concepts. *Papers from the eighth regional meeting, Chicago Linguistic society.* Chicago: University of Chicago Linguistics Department.

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

Lindenberg, S. (2001). Intrinsic motivation in a new light. *Kyklos, 54(2/3), 317-342.*

Lord, R.G., Brown, D.J., & Freiberg, S.J. (1999). Understanding the dynamics of leadership: the role of follower self-concepts in the leader/follower relationship. *Organizational Behavior and Human Decision Processes, 78, 1-37.*

Lord, R.G., Brown, D.J., Harvey, J.L., & Hall, R.J. (2001). Contextual constraints on prototype generation and their multilevel consequences for leadership perceptions. *Leadership Quarterly, 12(3), 311-338.*

Lord, R.G., Foti, R.J. & DeVader, C.L. (1984). A test of leadership categorization theory: internal structure, information processing, and leadership perceptions. *Organizational Behavior and Human Performance, 34, 343-378.*

Lord, R.G., Foti, R.J., & Phillips, J.S. (1982). A theory of leadership categorization. In J.G. Hunt, U. Sekaran, & C. Schriesheim (Eds.), *Leadership: beyond establishment views.* Carbondale, IL: Southern Illinois Univ. Press.

Lord, R.G., & Maher, K.J. (1991). *Leadership and information processing: Linking perceptions and performance.* Boston: Unwin Hyman.

Markus, H., & Wurf, E. (1987). The dynamic self-concept: a social psychological perspective. *Annual Review of Psychology, 38, 299-338.*

Meindl, J.R. (1993). Reinventing leadership: a radical, social psychological approach. In J. Keith Murnighan (Eds.), *Social psychology in organizations: advances in theory and research.* New Jersey: Prentice Hall.

Moon, J.Y., & Sproull, L. (2000). The Essence of the Distributed Work: The case of the Linux Kernel. Article available at http://www.firstmonday.org/issues/issue5_11/moon/index.html

Offermann, L.R., Kennedy, J.K., & Wirtz, P. W. (1994). Implicit leadership theories: content, structure, and generalizability. *Leadership Quarterly, 5(1), 43-58.*

Raymond, E. 1996. *The New Hacker Dictionary (3rd ed.)* Cambridge, MA: MIT Press.

Raymond, E. 1999. *The cathedral and the bazaar: musings on Linux and open source from an accidental revolutionary.* Sebastopol: CA: O'Reilly and Associates.

Rosch, E. (1975) Cognitive representations of semantic categories. *Journal of Experimental Psychology: General., 104, 192-233.*

Rosch, E. (1978). Principles of categorization. In E. Rosch & B. B. Lloyd (Eds.), *Cognition and categorization.* Hillsdale, NJ: Erlbaum.

Rummel R.J., (1970). Understanding factor analysis. Article summary of Rummel's *Applied Factor Analysis*, available online at: http://www.hawaii.edu/powerkills/UFA.HTM

Rush, M.C., Thomas, J.C., & Lord, R.G. (1977). Implicit leadership theory: a potential threat to the internal validity of leader behavior questionnaires. *Organizational Behavior and Human Performance, 20, 93-110.*

Ryan, R.M., & Deci, E.L. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, 25, 54-67.

Saers, N., (2003). A project model for the FreeBSD project. Candidatus Scientiarum Thesis at the University of Oslo, Institute for Informatics, available at: http://niklas.saers.com/thesis/thesis.html

Smith, J.A., & Foti, R.J. (1998). A pattern approach to the study of leader emergence. *Leadership Quarterly, 9(2), 147-160.*

Stallman, R. (1999). The GNU Operating System and the Free Software Movement. In C. DiBona, S. Ockman and M Stone (Eds.), *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA: O'Reilly.

Von Hippel, E. (1994). “Sticky information" and the locus of problem solving: implications for innovation. *Management Science, 40(4), 429-439.*

Von Hippel, E. (2002). Horizontal innovation networks – by and for users. MIT Sloan School of Management paper available for download at: http://opensource.mit.edu/papers/vonhippel3.pdf

Wittgenstein, L. (1953). *Philosophical investigations.* New York: Macmillan.

Yukl, G. (2002). *Leadership in organizations (5th ed.).* Upper Saddle River: Prentice Hall.