

**An Analysis of the Dynamics of the Legitimation Processes of Innovations in
Open Source Software:**

A Qualitative Study of the Rational Deliberations in the Drupal Project

By Soran Nouri

A Thesis Presented to Ryerson University
in Partial Fulfillment of the Requirements for the Degree of
Masters of Management Science (MMSc)
For the Program of Management of Technology and Innovation
In the Ted Rogers School of Management
Toronto, Ontario, Canada, 2013

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis may be made electronically available to the public.

Abstract

Analyzing the Innovation Legitimation Processes in Open Source Software:

A Deductive Qualitative Study on the Rational Deliberations of the Drupal Community

For the Master of Management Science (MMSc) Program

Management of Technology and Innovation

Ryerson University

Soran Nouri

Within the Open Source Software (OSS) literature, there is a lack of studies addressing the legitimation processes of innovations that are born in OSS. This study sets out to analyze the legitimation processes of innovations within the deliberations of the Drupal project. The data set constitutes 52 rational deliberation cases discussing innovations that were proposed by members of the community. Habermas's Ideal Speech Situations (ISS) is used as the framework to view Drupal's rational deliberations from; in fact within the 52 cases that are examined in this thesis, there were no violations to the guidelines of the ISS in the deliberations. The Communicative Action Theory, Influence Tactics theory and the theory of Validity Claims are aspects of the framework that is used to code and analyze the conversations. These aspects allow for an effective conceptualization of the dynamics of the Drupal deliberations. This thesis was able to find that legitimation processes of innovations in open source software were influenced by the type, complexity and implications of the innovations on the rest of the community. Also, bug fixes, complex innovations and innovations that have implications on the rest of the software will result in a long (in terms of number of comments) legitimation process. Also, it is empirically backed in this study that in open deliberations that aim at achieving mutual understanding towards a common goal, the communicative action type and the rational persuasion influence tactic are the most common methods for innovators to interact with the community.

Acknowledgement

I express my sincere gratitude to Dr. Ojelanki Ngwenyama, who was not only influential in the development of this thesis, but also in my learning experience during my enrollment in Ryerson University. I am also grateful for the study space that Dr. Ngwenyama's institute provided me, and to Ryerson University for providing that institute with research space. Also, I am thankful for all who are responsible and hardworking for the operations of the Ted Rogers School of Management. This thesis research was supported by SSHRC Grant #410-735213; I am grateful for this support, without it I could not have completed my degree studies.

Contents

1. Introduction.....	7
1.1 The Scope of the Study	8
1.2 Research Goals.....	8
1.3 The Research Approach	8
1.4 Structure of the Thesis	9
2. Literature Review.....	10
2.1 Background	10
2.2 Key Themes in OSS Research	13
2.2.1 The Political Economy of OSS	13
2.2.2 Literature on Intellectual Property Rights and Innovation in OSS	15
2.2.3 The Organization of OSS Development	18
2.3 Legitimation of IT Innovations	21
3. Theoretical Framework	24
3.1 Situated Rational Deliberations	25
3.2 The Social and Organizational Context	26
3.2.1 Social and Organizational Structures	27
3.3 Social Actions and Influence Tactics.....	28
3.3.1 Communicative Action Types	28
3.3.2 Validity Claims	29
3.3.3 Influence Tactics	31
4. Organizational Context of the Case	32
4.1 Background.....	32
4.2 Design and Technical Information.....	33
4.3 Social and Organizational Structure	34
5. Research Methodology	36
5.1 Empirical Analysis Procedure.....	36
5.2 Summary of Empirical Analysis	37
6. Discussion of Empirical Findings	46
6.1 Basic Characteristics of Uncommitted Innovation Proposals	47
6.1.1 Communicative Characteristics of SRDs.....	48
6.1.2 Examples of SRDs of Uncommitted Innovation Proposals	49

6.1.3	Selective Illustration of the SRD Dynamics of Case 2	50
6.2	Basic Characteristics of Committed Innovation Proposals	57
6.2.1	Communicative Characteristics of SRDs	58
6.2.2	Examples of SRDs of Fast Commits.....	59
6.2.3	Illustrative Examples.....	60
6.2.4	An Example of a Slow Commit SRD	65
6.3	Concluding Summary	74
7.	Theoretical Discussion.....	75
7.1	The Social and Organizational Context of Drupal.....	75
7.2	Leadership, Roles and Rewards	76
7.3	The Dynamics of Structured Deliberations.....	79
7.4	Communication Dynamics of Structured Deliberations	80
7.5	Characteristics of Legitimation Processes	82
7.6	Use of influence Tactics in Structured Deliberations	84
8.	Conclusion	88
8.1	Implications of this Thesis	89
8.2	Limitation.....	89
8.3	Recommendation for Future Research.....	90
	Bibliography	91
	APPENDICES.....	100
	APPENDIX A.....	100
	APPENDIX B	101
	APPENDIX C	102
	APPENDIX D.....	105
	APPENDIX E	108
	APPENDIX F.....	109
	Reflection.....	109

List of Figures

Figure 3-1 The Conceptual Structure of the Ideal Speech Situation.....	24
Figure 3-2 Explaining the Different Social Action Types	29
Figure 5-1 Summary of social action types for 24 committed cases	39
Figure 5-2 Summary of social action types for 28 uncommitted cases	40
Figure 5-3 Summary of Influence Tactics in 24 Committed Cases	40
Figure 5-4 Summary of Influence Tactics in 28 Uncommitted Cases	40
Figure 5-5 Summary of the Validity Claims Challenged in 24 Committed Cases	41
Figure 5-6 Summary of Validity Claims Challenged for 28 Uncommitted Cases.....	42
Figure 5-7 Summary of Types of Innovation Proposals for 24 Committed Cases	42
Figure 5-8 Summary of Types of Innovation Proposals for 28 Uncommitted Cases	42
Figure 6-1 Basic Characteristics of 28 Uncommitted Innovation Proposals	47
Figure 6-2 Communicative Characteristics of the Uncommitted Innovation Proposals.....	48
Figure 6-3 Communicative Characteristics of Two Uncommitted Examples	49
Figure 6-4 Summary of Basic Characteristics of the Case of Innovation Proposals	57
Figure 6-5 Comparison of the Communicative Characteristics of Fast and Slow Commits	59
Figure 6-6 Summary of Characteristics of the Three Example Fast Commit Cases.....	60
Figure 6-7 Summary of Communicative Characteristics of the Three Example Fast Commit Cases	61
Figure 6-8 Communicative Characteristics of Slow Commit Case 12	65

1. Introduction

The Open Source Software Development (OSSD) methodology is an emergent and rapidly growing methodology (von Hippel & von Krogh, 2003; Lerner & Tirole, 2001). It started with the General Public License (GPL) initiative by Richard Stallman, who created the GPL initiative as a response to a certain source code that he was using becoming private (Vainio & Vaden , 2007). In turn and as an act of defiance to the privatization of formerly public source code, Stallman invented the GPL, which serves to ensure that software under its license is accessible and distributed free of charge. The GPL is also used by many other successful and popular open-source projects such as Linux, Apache, Google, Android, etc. (Mustonen, 2003). Such software brought attention to the open-source methodology, a software production methodology that is proving worthy of challenging the traditional and hierarchal nature of software production; in fact there is an abundance of evidence within the literature proving so (Bonaccorsi & Rossi, 2003; Crowston, Annabi, & Howison, 2003; Kogut & Metiu, 2001; Strang & Macy, In Search of Excellence: Fads, Success Stories, and Adaptive , 2001). This growing popularity has led to an increase in studies that are aimed at understanding the open source phenomenon and the way in which it operates. Some key reasons for this popularity are: (1) the importance of open source software to social and economic sectors; (2) interest in understanding the economic and social organization of open source software development and (3) interest in understanding how open source projects drive innovation.

There has been a range of studies on the economic paradox of open source software projects (Zeitlyn, 2003; Tirole & Lerner, 2002; Ashton & Oakley, 1997), and the social composition of the open source community, which is entirely based on a voluntary and self-organizing system (Ljungberg, 2000; Lakhani & von Hippel , 2003; Crowston & Scozzi, 2003; Crowston, K, Wei, Eseryel, & Howison, 2007). However, there are still outstanding areas of research worth addressing, particularly the lack of studies that focus on the legitimation processes of the innovations that are essential to OSSD. This thesis is an attempt to fill this gap in the literature.

1.1 The Scope of the Study

This study will examine the dynamics of legitimizing innovations in open source software by analyzing conversations of the Drupal open source web content management system. These conversations discuss Drupal's operations, and thus entail information that aids this study in understanding the underlying dynamics of how the open source project innovates. This study will attempt to explain dynamics of the rational deliberations in Drupal from the theoretical perspective of the Ideal Speech Situation (ISS) (Habermas, 1970). Three aspects of the framework are used to help me code and analyze the dynamics of legitimizing innovations. These aspects are: (1) Habermas's theory of Communicative Action (CAT), (2) the theory of Validity Claims (VC) and (3) the Influence Tactics (IT) theory.

1.2 Research Goals

In this thesis I set out to address the following research question: What are the dynamics of the innovation legitimation processes in open source software development? To answer this question this study entails a critical discourse analysis to examine the legitimation processes of the innovations. This research is relevant and persisting because although interest for open source projects has seen much growth, there is a lack of research on what is potentially OSSD's most promising aspect, their ability to innovate. Thus, before any insight from OSSD innovations can be transferred onto other disciplines, we must first examine and understand its dynamics in OSSD.

1.3 The Research Approach

This is an explanatory study; it aims at explaining the dynamics of the legitimation processes of innovations in open source software. This thesis uses a deductive approach to the research; it utilizes the Ideal Speech Situation (ISS) (Ngwenyama O. , 1993; Ngwenyama & Lyytinen, 1997) as the conceptual framework and primary strategy for analyzing the rational deliberations of the Drupal development team, and uses the Critical Discourse Analysis (CDA) method developed by (Cukier, Ngwenyama, Bauer, & Middleton, 2009). This method offers a strategy and a set of procedures for interrogating discourses to identify empirical observations concerning validity claims which are embedded in conversations. The Drupal conversations at hand are viewed by

this study as Situated Rational Deliberations that are coded and analyzed by the CAT, IT, and VC.

This study follows the interpretivist approach. This is appropriate because I interpret the content of the deliberations and code accordingly. The empirical materials for this study comprises a corpus of 6,000 pages of textual data documenting conversations of the Drupal open source development community between 2003 and 2004. The data analysis was qualitative in nature, and was conducted using HyperResearch, a qualitative analysis software.

1.4 Structure of the Thesis

This thesis is built in a series of chapters. Chapter Two discusses key themes of the OSSD literature, with a focus on innovation and legitimation in open source software projects and the difference between the traditional and open source structures. Chapter Three elaborates on the theoretical framework that is used in this thesis. Chapter Four situates the context of the case, explaining the organizational and social context of the Drupal project. Chapter Five discusses the research methodology, where the approach to the data collection, coding and analysis are discussed. Chapter 6 discusses the data analysis, Chapter Seven is the theoretical discussion, where the data analysis is corroborated with the literature and propositions are made. Lastly, Chapter 9 is the conclusion of the thesis, with remarks on the limitations of the study and potential future research.

2. Literature Review

2.1 Background

The birth of open source software development (OSSD) was attributed to a group of computer engineers and scientists who considered the free sharing of software code an important aspect of their knowledge development and collaborations. A common practice within the academic culture is the sharing of information and material serving the purposes of collaboration and corroboration, in comparison, it is fitting to believe that scientists, engineers, and other contributors within the software development industry would also support such open source initiatives, and are willing to share code for the sake of “better code”. After all, the collaborative attitude of the scientific community is one of its key success factors that propelled its status to the level of knowledge and advancement seen today (Krogh, 2003; Terttu Luukkonen, 1992; Olle Persson, 2004).

In 1969, the OSSD initiatives received a strong boost in operation when the United States’ Defense Advanced Research Project Agency (DARPA) established the ARPAnet, which was “the first transcontinental, high-speed computer network. ARPAnet allowed developers to exchange software code and other information widely, easily, swiftly and cheaply” (Krogh, 2003; Clark, 1988; leiner, et al., 1997). With the rapid advancement in contributions to this field, ARPAnet soon became overloaded; and a bottleneck was formed due to the large processing demand. This bottleneck would only be solved by the creation of the Internet. The Internet, and the technologies that allow for it to function such as web servers and routing protocols, are responsible for boosting the number of hosts handled by ARPAnet from 250 hosts, to more than One Million users worldwide, as of Spring of 2003 (Krogh, 2003; leiner, et al., 1997).

The Start of Open Source Software

MITs artificial intelligence lab had a significant role in the birth of the Open Source Software (OSS) concept. The lab had a communal culture present among a group of programmers since the 1960s. The OSS movement all started in the 1980s, when MIT licensed a piece of software to a commercial software firm, which later restricted access to its source code, blocking even the programmers at MIT who helped develop the software from accessing it. To the software

development team at MIT that developed this piece of software, this meant that they would no longer be legally allowed to either use or build upon this software. As a reaction, Richard Stallman, an accomplished programmer at MIT, founded the Free Software Foundation (FSF), which sought to diffuse a legal initiative that would maintain the programmers' right to access software they helped write. "The basic license developed by Stallman, in order to implement this idea, was the General Public License or GPL. The basic rights transferred to those possessing a copy of free software included the right to use it at no cost, the right to study and modify its 'source code', and the right to distribute modified or unmodified versions to others at no cost" (Krogh, 2003; Vainio & Vaden , 2007; Carver, 2005).

Later, the term 'Free' software appeared troublesome for commercial software developers. So, in response to this notion, Bruce Perens and Eric Raymond, along with other prominent hackers founded what is now known as the *Open Source* movement. Open source licensing covered the same licensing practices as the initial *Free* Software Foundation movement. But unlike the *Free* software movement that requires the software to be posted entirely in the public domain, Open Source Definition requires the software's source code as well as compiled form to be distributed (Kogut & Metiu, 2001; Vainio & Vaden , 2007). This license may not require a royalty or purchase price, but it allows for developers to gain monetary return from variations of software that they have modified "without distributing the source code of the modification" (Kogut & Metiu, 2001; Vainio & Vaden , 2007). For example, The Berkeley System Distribution (BSD) and the Apache web server allow for private sale of a modified version of the software, without the distribution of the source code. But the original version of the software will still be attainable for no cost. (Kogut & Metiu, 2001; Vainio & Vaden , 2007).

Table 2-1 below identifies a list of OSS projects that became popular in the recent years, and that are now used on a frequent basis in the commercial industry (Kogut & Metiu, 2001).

Table 2-1 Examples of Open Source Projects

Name	Definition/description
Linux	An open-source server operating system
Apache	An open-source web server
Zope	Enables teams to collaborate in the creation and management of dynamic web-based business applications such as intranets and portals
Sendmail	The most important and widely used e-mail transport software on the Internet
Mozilla	Netscape-based, open-source browser
MySQL	Open-source database
<u>Scripting Languages:</u>	
Perl	The most popular web programming language
Python	An interpreted, interactive, object-oriented programming language
PHP	A server-side HTML embedded scripting language
<u>Other:</u>	
Bind	Provides the domain-name service for the entire Internet

Adopted from: (Kogut & Metiu, 2001)

The significance of the Open Source Software Development (OSSD) methodology becomes relevant when the various successful open source projects within the software development literature are reviewed. The OSSD processes vary from the Traditional Software Development (TSD) processes in ways that will be discussed below. Some Open Source Software (OSS) that have been in use by companies such as IBM, NASA, and governments such as the German government are a testament to the notion that regardless of the methodology used to develop software, efficient software is in demand (Krogh, 2003; Mockus, Fielding, & Herbsleb, 2002). Popular OSS projects such as the projects outlined in table 1 are proof that OSS is relevant and persisting in the software development industry.

2.2 Key Themes in OSS Research

Research on open source software development has been largely dominated by three important themes: (1) The political economy of the approach; (2) Innovation and intellectual property rights; and (3) the organization and management of open source software development projects. However, more recently some new questions about the viability of open source software are asked. In this chapter I will provide an overview of the three themes of OSS, and based on this literature review, I will then proceed to situate my thesis topic within the literature.

2.2.1 The Political Economy of OSS

Open source software development is largely a *Gift Economy* in which “thousands of top-notch programmers contribute freely to the provision of a public good” (Krogh, 2003). Such an economic organization is in contradiction to the dominant logic of capitalist economics which argues that human beings are self-interested agents who seek only to maximize their utility/earnings, and from that capitalist economic perspective, OSSD is an interesting paradox (Kogut & Metiu, 2001; Bitzer, Schrettl, & Schroder, 2007; Hertel, Niedner, & Herrmann, 2003). The Oxford Review of Economic Policy journal states that “the natural resolutions to this paradox are to tie provisions to intrinsic reward or to supplementary extrinsic rewards” (Kogut & Metiu, 2001).

While the reasons as to why top-notch programmers dedicate their time, for no direct financial return, to OSS initiatives have sufficiently been discussed in the OSSD literature, the ability by the OSS movement of stopping the commercial industry from gaining financial returns from advancements in OSS is still an area of concern. (Kogut & Metiu, 2001; Bitzer, Schrettl, & Schroder, 2007).

These perspectives provide a solid reason as to why OSS has gained traction within the last two decades, there is also an emerging trend in the software development discourse supporting the efficiency of OSSD, and challenges “the theory of the second best”, which states that “innovators will not innovate if they do not have patent protection” (Kogut & Metiu, 2001). It has been apparent that programmers are enrolling in OSS for intrinsic and supplementary extrinsic

rewards. An intrinsic reward is the self-satisfaction from doing public good. Extrinsic reward is the potential to increase chances in the labor market via an enhanced reputation that a programmer attains by enrolling his/her name with a popular, successful OSS project. (Kogut & Metiu, 2001; Hertel, Niedner, & Herrmann, 2003).

According to von Hippel and von Krogh (2003)

Open-source software developers freely reveal and share because they garner personal benefits from doing so, such as learning to develop complex software, perfecting expertise with a computer language, enhancing their reputation, and for pure fun and enjoyment. Many of these benefits depend on membership in a well-functioning developer community.

OSS developers also find it beneficial to enroll in the development of OSS because feedback is received from other top-notch programmers. This can be valuable for both the expertise of the contributing programmer as well as for the software being produced, especially if the software is of complex nature. One perspective claims that OSS developers are encouraged by 'Altruism'. For example, Kahneman (1986) and Bies (1993) point out in their studies that OSS developers actually ignore economic calculations when it came to contributing their time to OSS projects. This same study found that OSS developers sometimes shared a fixed reward, also developers defected less from OSS projects that included more communications (Kahneman, Knetsch, & Thaler, 1986; Bies, Tripp, & Neale, 1993).

On the other hand, Lerner and Tirole (2001) argue that the reasoning behind the free contributions of the OSS community is that it does not take much time for programmers to contribute code since these expert programmers have usually already written/modified codes for their private applications. Lerner and Tirole (2001) also state that the code is not always written for the sole purpose of sharing with the rest of the community, rather it is written for personal applications, but is then turned over to the community in hopes of further improving the code, which can then further benefit the programmer who initially submitted the code. In addition, Lerner and Tirole (2001) present the notion that developers gain merit by submitting efficient code to the community. Thus, contributions to high-profile OSS projects can potentially increase chances in the labour market.

In another study conducted on the Apache web server support groups, it is apparent that some OSS developers operate under a reciprocal system, in which the developers' desire to help other developers stems from their appreciation for having been helped by other programmers in the past, or for the purpose of paying it forward for when the need be in the future (Lakhani & von Hippel, 2003).

In another study by Hertel et. al (2003) on the Linux operating system kernel, it was concluded that the positive image and the fact that the software is well known creates a sense of responsibility from developers submitting code towards the software. The group dynamics also motivated the contributing programmers by instilling in them a sense of indispensability. In addition, it seems that programmers contribute to OSS due to personal motives of producing better quality software of their own. In other words, they want the opportunity to learn certain areas of knowledge, which they probably would not otherwise learn, and to be able to apply that knowledge to their personal work.

Commercial software development projects usually put programmers under a contract, which encompasses various aspects of the production cycle such as structure, style, due date and other such restrictions. On the other hand, programmers that volunteer to enroll in an OSS project are under no legally binding contract to contribute, in addition they have the flexibility as to what, when and how to contribute. From the perspective of the traditional software development (TSD) methodology, this can be found counterproductive as there is a lack of control by management on the contributing programmers, however an alternate school of thought views the OSSD methodology as a productive one.

2.2.2 Literature on Intellectual Property Rights and Innovation in OSS

Intellectual Property Rights

In the commercial industry, intellectual property law guarantees programmers, or the organizations that they work in, revenue. In contrast, intellectual property rights in the OSS industry guarantees future users against appropriation for using the software (Vainio & Vaden, 2007; Dahlander, 2005; Mustonen, 2003; Bonaccorsi & Rossi, 2003).

A common aspect of the various open source licensing practices is that the intellectual property rights to the software are in the public domain. However, due to an emerging trend of software developers who believe that a hybrid between public and private intellectual property rights present in the same software results in better software, discussions about variations between OSSD methodologies have started, with the main varying factor between different open source licenses being the extent to which private intellectual property rights are to be included within an OSS (Osterloh & Rota, 2007; Tirole & Lerner, 2002).

Under the General Public License (GPL), a developer can make money from the software, and that same software could also be available at no cost, which means the software could be double licensed (Vainio & Vaden, 2007; Osterloh & Rota, 2007; Lerner & Tirole, 2001). According to Kogut and Metiu (2001), “the boundaries between the public and private segments of the software developed by the open-source community are thus not distinct”. Therefore, as the OSSD communities further advance, this is one aspect that will need to be addressed more clearly and concisely, in order to stimulate further interaction with the commercial industry.

In essence, the OSS industry seeks to protect the software from being hidden and privatized. This GPL license allows for incremental innovation from developers in the community who collectively offer the potential of effectively fixing the code (Bonaccorsi & Rossi, 2003).

Innovation in Open Source Software

In essence, an invention is a scientific breakthrough, and an innovation is the commercialization of that breakthrough (Schumpeter, 1934; Roberts, 2007; Nelson & Winter, 1982). Moreover, innovation that occurs within an open source environment is ‘Open Innovation’. According to Gallagher and West (2006), open Innovation is a result of a firm’s use of a broader range of sources than the traditional ‘Vertically Integrated’ approach to innovation. For example, a firm fostering open innovation can utilize its customers, competitors, firms in other industries, and academics in its innovative activities. Essentially, Gallagher and West (2006) define open innovation as:

“systematically encouraging and exploring a wide range of internal and external sources for innovative opportunities, consciously integrating that exploration with firm capabilities and resources, and broadly exploiting those opportunities through multiple channels”

Since open innovation takes into consideration external sources of innovation, such as leveraging the research of others, Gallagher and West (2006) identify a few issues that firms should address in order to encourage open innovation. Firstly, like traditional approaches to innovation, internal innovation is still a significant source of innovation, and firms must optimize its use of internal Research and Development (R&D) to maximize its capabilities. Such capabilities will allow for innovations to be internally and externally commercialized, and can also potentially increase the firm’s capability of recognizing and utilizing innovations beyond the firm’s boundaries (external innovation) (Gallagher & West, 2006). Secondly, Gallagher and West (2006) recommend taking advantage of external sources of innovation by commercializing innovations from other firms or industries, consulting with clients and suppliers, and utilizing governmental and academic research. The challenge here is to identify relevant external innovation to spend the firm’s resources on. Lastly, Gallagher and West (2006) address a situation where external innovation becomes scarce, due to the over-dependence of firms on governmental and academic sources of innovation, or on each other. Thus, it is recommended that incentives, whether intrinsic or extrinsic, be given towards internal innovation to promote the spillover of innovation. Innovation being ‘spilled over’ by firms in an industry can potentially increase the amount of innovation for firms within such industry.

In comparison with patents laws and private intellectual property rights, Mazzoleni and Nelson (1998) state that public intellectual property rights deter innovators from innovating because they cannot own the rights to the innovation, and are thus not guaranteed monopolistic profit from the innovation. On the other hand, according to (Kogut & Metiu, 2001) “the strong recent expansion of the legal protection of software from copyright to patent has been decried as a threat to innovation and to the sharing of knowledge in fast-paced industries”. For example, Lerner (1995) found that patents by large firms in bio technology have effectively deterred smaller firms from innovating in this field. In other words, the patenting system has defeated its own purpose of

providing an incentive to innovate. According to David (2000), this threat is also present in the academic research field.

2.2.3 The Organization of OSS Development

The rapid growth of OSSD suggests that it is a relevant methodology of software production (Bonaccorsi & Rossi, 2003). This brings into focus the argument of Kogut and Metiu (2001), which state that TSD only continues to enjoy success due to the commercial industry's desire to profit from intellectual property rights. In other words, once profit is put aside, TSD is deemed a less efficient software production methodology in comparison to OSSD, and although it is still an object of controversy, the success of OSS in recent years leaves little doubt that it is a convincingly appealing methodology for software development (Tirole & Lerner, 2002; Mockus, Fielding, & Herbsleb, 2002). In addition to challenging the 'Second best' theory, OSSD's ability to concurrently design and test software is perceived as an advantage over the highly bureaucratic and structured approach of TSD. This concurrency in designing and testing is achieved by utilizing more effective communication methods via the Internet (Mockus, Fielding, & Herbsleb, 2002; Bonaccorsi & Rossi, 2003).

The dynamics of the growth of OSSD are reminiscent of those of the Internet. The internet was started by the US army, and further developed by federal programs to facilitate communication among researchers. Essentially, the internet started on the notion of disseminating ideas for the purpose of distributed collaboration, but the Internet itself is also a result of interconnected networks and worldwide collaboration, just as the OSSD industry operates nowadays. "The world wide web is an open source software program" (Kogut & Metiu, 2001). Similarly, the production of academic research is also bound by "strong norms regarding the public ownership of knowledge and the importance of public validation of scientific results" (Kogut & Metiu, 2001). Although there is a bold line between the scientific community's stance on public access and the commercial industry's focus on property rights, when the popularity of the internet and advances of the scientific communities are brought into perspective, OSSD appears more promising than it already has proven to be and less puzzling to skeptics (Kogut & Metiu, 2001; Crowston, Annabi, & Howison, 2003).

Open source software development is mostly organized as flexible, virtual and collaborative teams (Bergquist & Ljungberg, 2001; Crowston & Scozzi, 2003). While collaborators in an OSS project may vary in culture and geographic location, they are highly motivated volunteers who share an interest in the realisation of the proposed software product and are committed to the values of the global OSS community (Gallivan M. J., 2001; Stewart & Gosain, 2006; Bergquist & Ljungberg, 2001). OSS projects often start with an entrepreneurial developer who has an idea for a software product which he/she thinks might interest the OSS community. The entrepreneur then ‘shops around’ the idea and recruits volunteers from the many OSS online communities to collaborate in developing the software product. OSS initiatives find it relatively easy to attract highly motivated and competent collaborators to carry out the software project (Krogh, 2003). The flexibility of a virtual organization, voluntary participation, high motivation and solidarity among the developers enable highly effective OSS teams (Koch & Schneider, 2002; Gallivan M. J., 2001). Participants of OSS projects find that this organizational environment offers them more freedom to be creative, improve their skills and produce higher quality software than traditional commercial development projects do (Krogh, 2003; Koch & Schneider, 2002; Kogut & Metiu, 2001). While some volunteers are simply intrinsically motivated, others view OSSD projects as opportunities to polish their skills and gain a reputation which can be leveraged in the wider software labour market.

2.2.3.1 *Intrinsic reward versus extrinsic reward*

Also, Kogut and Metiu (2001) make an interesting analogy by comparing software development with Richard Titmuss’s study on blood donors (Ashton & Oakley, 1997). In his book, Titmuss claims that when blood donors are intrinsically rewarded, meaning they are motivated by the public good and blood is donated as opposed to sold, the quality of blood remains high. However, when blood donors are extrinsically rewarded, meaning they are paid for their blood, the results are potentially disastrous for the blood market. Titmuss came from the school of thought that stated that blood donors themselves best know the quality of their blood, and that a blood donor that is intrinsically motivated would only donate blood if it is for the public good, however an extrinsically motivated blood donor would donate for the purpose of monetary return regardless of the quality of the blood being donated, and thus an extrinsically motivated blood donor would be less hesitant to donate low quality blood. Titmuss believed that extrinsically

motivated blood donors were a threat to the blood market because the need to filter the donated blood would be overly expensive for the blood market, thus potentially crashing it. In other words, “a voluntary policy provides a highly motivated donor” (Kogut & Metiu, 2001). This analogy is familiar to the quality of software between OSSD and TSD.

2.2.3.2 *OSSD versus TSD Processes*

In an attempt to optimize the process of producing software, the commercial software industry adopted what Cusumano (1991) calls a ‘Software Factory’ approach, in which the software production is routinized. “This approach culminated in an attempt to rationalize the entire cycle of software production, installation, and maintenance through the establishment of factory-like procedures and processes.” Kogut and Metiu (2001) indicate however, the factory-like approach is not always recommended for high quality and innovative software production. Glass (2006) views the software production process as a creative one, and believes that attempts to turn the processes of software production including design, testing, coding, etc., into a tightly structured format will have negative effects on the software. Namely, such an approach will increase the complexity of the software, and make it difficult for detailed-level problems to be resolved.

According to Subramanyam and Krishnan (2003), extreme complexity of software is software’s worst enemy. Therefore, the software development literature had pointed towards initiatives that would help reduce the complexity of software development, such initiative is *hidden knowledge* within different modules of the software (Kogut & Metiu, 2001). The use of modules allows for greater control over the aspects of that one module, and also allows for a narrower scope of accountability for that one module, and thus better management of it. Another initiative to reduce software complexity is the partitioning of different development processes to be managed under different team leads. These two development concepts of ‘hidden knowledge’ and partitioning of development processes that help reduce complexity of software is commonly utilized by OSSD. In addition, OSSD enjoys the advantage of a more diverse and distributed community than commercial software development teams do, made possible by the use of the Internet (Vainio & Vaden , 2007; Crowston, Annabi, & Howison, 2003; Bonaccorsi & Rossi, 2003). The diverse community of developers in OSSD also allow for a more efficient

development process because design and debugging can be done concurrently. This is because when the source code of an OSS is released, the code is debugged on a decentralized basis.

Also, OSSD utilizes a more flexible and interactive approach. This is done by the use of modular design, which allows for utilizing the intellectual property of the masses, coupled with the software license that allows for individual developers to modify the module of their choice. Allowing programmers to contribute in the module of their choice provides better fit between task and competency (Kogut & Metiu, 2001; Vainio & Vaden, 2007).

2.3 Legitimation of IT Innovations

According to Kaganer et. al (2010), most of the definitions of legitimation in the organization literature fall under one of two categories: Strategic or Institutional. The strategic approach depicts legitimacy as an organizational tool used to accomplish goals, while the institutional approach considers legitimation as a perceived acceptance of an innovation, based on the value system of the institution (Scott, 2001; Ashforth & Gibbs, 1990; Pfeffer, 1981). Recent literature has attempted to merge the two approaches with the rationale that although organizational environments and their underlying values and beliefs are constitutive of legitimation, organizational actors do have the capacity to strategize the legitimation of IT innovations based on goals (Suchman, 1995; Golant & Sillince, 2007; Oliver, 1991).

Traditionally, legitimation is believed to be controlled by decision makers, executives or key stakeholders in an organization, or what Cyert and March (1963) name a “dominant coalition”. According to Cyert and March (1963), “A dominant coalition consists of the network of individuals within and around an organization that most influence the mission and goals of the organization. One role of dominant coalitions within organizations is to create a system in which they encourage compliance from organizational members towards a laid out strategy (Cyert & March, 1963; Pattigrew, 1985). This notion of dominant coalitions corresponds to one of the traditional legitimation perspectives in the literature: the ‘Strategy-Legitimation Nexus’, which is discussed from a multitude of perspectives in the Management discourse, with the ‘Implicit message ... that organizational leaders have near monopolistic control over the interpretive process and that they determine the limits to their control’ (Neilson & Rao, 1987). For example, Pondy (1978) views leadership as a ‘language game’, where the leader is the ‘linguist, and

mentor who shapes the values and frames the organizational members`. Kramer (1975) illustrates that some dominant coalitions use `planning systems and analyses to legitimate political choices, clothing the choice process with a veneer of value free rationality`. Also, Salancik and Meindl (1984) exposed the notion that dominant coalitions take responsibility for negative outcomes in an attempt to influence organizational members into perceiving that the coalition possess control over strategy in a `hostile` environment.

While significant progress has been made in IT Innovation in the past 20 years, most studies were approached from the “Dominant paradigm of IT innovation research”, as Fichman (2004) labels it. This approach considers the organization to be the unit of analysis, and the notion that the adopters of new IT innovations adopt on a rationalistic basis (Fichman, 2004; Strang & Macy, 2001). However, a number of authors have pointed to over-rationalization of models that are outputted from the dominant paradigm, namely, due to the lack of focus on the technical and institutional aspects of modern organizations (Abrahamson, 1991; Currie & Parikh, 2005; Strang & Soule, 1998).

In an effort to account for “the complexity of today’s IT innovations and the degree of interconnectedness among potential adopters and other stakeholders”, Kaganer et. al (2010) built a model that takes into consideration the dynamics of the institution and the environment in which an innovation is to be diffused within. To accomplish this, Kaganer et. al (2010) adapted and extended the Organizing Vision framework from Swanson and Ramiller (1997), which distanced itself further from the dominant paradigm, and closer to an open discourse, by considering a “community discourse” in attempting to understand the application of a proposed innovation, as opposed limiting the evaluation of an innovation to a dominant coalition.

Legitimation Within Open Source Environments

As previously mentioned, the dominant figures and decision makers in open source projects come in different forms: Some projects are led by single actors, such as Dries and Torvalds in the Drupal and Linux projects, respectively, where they both possess the ultimate authority over their respective projects, while in the Apache project, suggestions from selected people will be considered based on merit. These developers are known as the Apache Board and are the only people that are allowed to make changes to the web server.

Regardless of the authority structure employed in an open source project, and as opposed to the monopolistic approaches mentioned in the dominant paradigm of IT innovation research, legitimization in open source projects can be better understood from the interpretive approach in the Information Systems literature that states that legitimacy in organizations is reached based on the input of a set of actors from different hierarchical levels within an organization (Bernard, 1938; Neilson & Rao, 1987). ``It is suggested that legitimation is a multilayered process of social discourse and that the message that dominant coalitions have unilateral control over interpretive processes is as questionable as the notion that authority is granted from above rather than consented to from those below in the organization hierarchy.`` (Neilson & Rao, 1987). Further explaining this notion, Neilson and Rao (1987) state that ``It is contended that the shared meanings that guide human behavior are rooted in the combined thinking of human actors and that the emergence and legitimation of these meanings involves complex interactions among all who have an effect on organizational functioning``.

3. Theoretical Framework

The theoretical framework for this research is rooted in the theory of communicative action (Habermas, 1970; Habermas, 1975; Habermas, 1989). Specifically, this research applies the theoretical concepts of the Ideal Speech Situation (ISS) to interrogate the deliberative discourses about software innovations in the context of open source software development. The ISS is concerned with social interaction in collaborative processes where the aim of the participants is to reach agreements for joint action via rational deliberation. According to Ngwenyama (1993) the ISS also provides “set rules and guidelines to facilitate fair and effective communication and rational discourse” upon which collaborative group processes such as software development could be designed and studied. The ISS theoretical framework offers a set of concepts which can be used to empirically interrogate the key characteristics and dynamics of deliberative discourses within both OSS and traditional software development projects. Figure 3-1 adopted from: Ngwenyama (1993) below illustrates the relationship of concepts of ISS framework that are relevant to this empirical study.

Figure 3-1 The Conceptual Structure of the Ideal Speech Situation

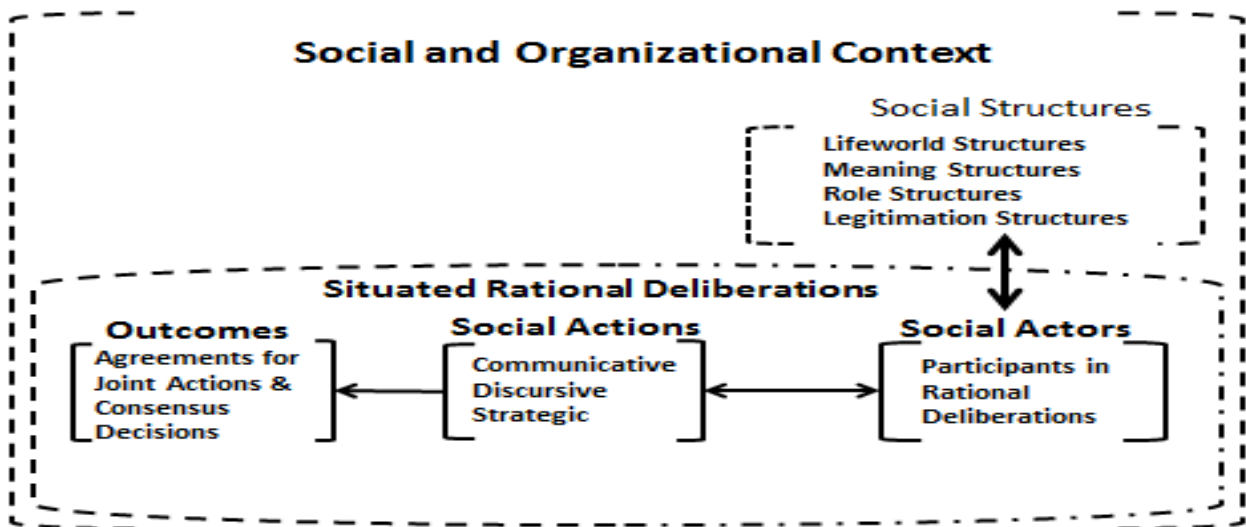


Figure 1: The Conceptual Structure of the Ideal Speech Situation

Adopted from (Ngwenyama & Lyytinen, 1997; Ngwenyama O. , 1993)

Habermas's theory of communicative action and the Ideal Speech Situation will be used to critically analyze data concerning the interactions of the actors in the Drupal open source community. Specifically, I am interested in interrogating the underlying structure and characteristics of the communication taking place among the actors with regards to the legitimization of proposed innovations. In this regard I will view these communicative interactions as a set of rational deliberations, the objective of which is to come to an agreement on which innovations to implement. It is important to note here that in the context of open source communities where work efforts are voluntary, implementations of new innovations must be agreed upon by key members of the community. These agreements are achieved via open debate (a set of Situated Rational Deliberations) in which the participants are free to state their opinions and to challenge and interrogate the opinions of each other. In this regard Habermas's theory of communicative action and the ISS is an appropriate framework for developing a theoretical understanding of the open debate of open source communities.

3.1 Situated Rational Deliberations

In this research I will view the open debate to legitimate software innovation proposals within the Drupal open source community as a set of *situated rational deliberations*. According to Ngwenyama (1993) the ISS offers a set of guidelines for studying the communicative actions within software development teams while the ISS sets out a simple set of rules to which participants must abide by during rational deliberations (Ngwenyama & Lyytinen, 1997). The goal of the ISS is to achieve agreements for collaborations, and the entire process falls within a social and organizational context which gives meaning to the rules Ngwenyama (1993). The rules of ISS are as follows:

1. All participants have equal status in the group deliberations
2. All participants have equal opportunity to raise issues, challenge, or defend the validity of all actions or statements.
3. All communications must be clear and understandable; no form of jargon may be used to mystify or erect barriers to communication
4. All statements must be relevant to the existing deliberations
5. All statements and actions must be appropriate to the situation under consideration.

6. All participants must say what they mean and take the action that is mutually agreed
7. All proposed actions will be meaningful and effective for achieving the intended goals

Some researchers Blake (1995) and Cooren (2000) have argued that no discourse can adhere to the guidelines of the Ideal Speech Situation as there are always time and space constraints. Also, if time was not a constraint, the process of reaching agreement would be inefficient, taking much longer than desired by the collaboration. These are however misunderstandings of Habermas's intent for the ISS. (Habermas, 1970) Used the term *ideal* in the theoretical and sociological sense, so did Weber (Burger, 1976). The term ideal is meant to identify a goal to aspire to with the full understanding that no social situation can fully attain it. The objective of Habermas's ISS is to provide a framework to help actors who aspire to free and open deliberations, avoid some of the common traps that undermine such deliberations.

3.2 The Social and Organizational Context

The social and organization context comprises norms, resources, incentive and recognition schemes that orient organizational activity. According to Gioia (1986) the social and organizational context embodies both explicit and visible structure and policies of the organization, as well as implicit norms and beliefs of the organization's lifeworld. As stated earlier, most OSSD is organized around virtual teams in which the participants are located in different geographic locations (Gallivan M. J., 2001; Stewart & Gosain, 2006; Bergquist & Ljungberg, 2001; Crowston & Scozzi, 2003). This often results in a set of team members with different cultural backgrounds carrying uniquely different ways of interpreting and acting in the world which often leads to breakdowns in understanding within the team (Olson & Olson, 2000; Malhotra & Majchrzak, 2004; Cramton, 2001; Bjorn & Hertzum, 2006; Malhotra & Majchrzak, 2004). Organizations and all forms of collaborative social actions require a set of social structures (shared beliefs and common organizational norms) to orient the actors (Bjorn & Ngwenyama, 2009).

3.2.1 Social and Organizational Structures

Three classes of social and organizational structures are relevant to my study: (1) structures that participants use to interpret and enact meaning in the actions of each other; (2) structures defining roles and responsibilities of the participants; and (3) structures that participants use to test the validity and legitimacy of their actions and interactions. Bjorn and Ngwenyama (2009) refer to these as lifeworld, organizational structures and work practices, respectively.

Lifeworld structures are the beliefs (social norms) and cultural meanings that guide people's behaviours and attitudes that are formed by lived experiences within certain social contexts (Habermas, 1985). While each participant brings to the team a unique lifeworld structure that includes cultural practices, meaning structures and rules of social interactions in addition to their technical competences, effective collaboration depends upon the team developing a shared set of social structures and a common context of meaning (Bjorn & Ngwenyama, 2009). Some researchers have argued that a common language, shared meaning and work practices are necessary for virtual organizations to function effectively (Olson & Olson, 2000; Malhotra & Majchrzak, 2004; Cramton, 2001; Bjorn & Hertzum, 2006). The lack of these contributes to the increase in the potential for breakdowns in communication (Ngwenyama O. , 1998; Bjorn & Ngwenyama, 2009).

Role structures define the different roles an actor can play and the duties and responsibilities for the role (Habermas, 1985; Bjorn & Ngwenyama, 2009). From the perspective of ISS, all actors have equal status in the rational deliberations and the moral obligation to fully engage in the debate while following the seven basic rules outlined above. Every deliberative process requires some rules of process and a stopping rule. In ISS the participants decide on the rules of process and the stopping rule, also known as the Obligatory Passage Point (OPP) in this study, before the deliberation commences. The OPP is defined at the start of the deliberation process; the participants discuss and come to agreement on what constitutes consensus agreement. Before the *rational deliberation* starts someone is appointed facilitator and monitor, and assumes the responsibility to keep the debate on course and to poll the participants for consensus. When

consensus is achieved (as defined by the participants) the deliberations are closed; the agreement for joint action are legitimated and the participants turn their attention on implementation.

It is important to note that while most of the rules of the deliberative process are articulated and available, most participants are socialized by participating in the deliberative process. In the case of an OSSD team, the more situated rational deliberations a new member witnesses, the more that new member realizes that members can state their opinions without restrictions. It is via active participation in the rational deliberations that new members are socialised into their right and obligations in the OSS community. This approach to organizational socialization is very different from a traditionally organized software development team. The ideals and lifeworld beliefs of an OSS community confer upon its members the moral obligation to extend their best efforts in the realization of the best software product that meets needs of the user community.

3.3 Social Actions and Influence Tactics

3.3.1 Communicative Action Types

In the ISS framework, (Habermas, 1974) outlines four basic types of communicative actions in which the participants of rational deliberations may engage (cf. Table 3.1). These are communicative, discursive, strategic and instrumental action (strategic action is beyond the scope of this thesis). In categorizing the actions of individuals within the social and organizational context Habermas acknowledges that individuals are more than just entities that are receptors of information, as the positivist perspective states, instead he recognizes individuals as “active persons or interpreters ... who create and enact the meaning that they come to hold” (Habermas, 1974). Also, in contrast to the interpretive perspective where the purpose of the communication is to achieve understanding, Habermas’s four types of social action address all forms of symbolic communication (Ngwenyama & Lee, 1997). According to the CAT, every utterance of an individual within a social context represents an intentional social action of one of the above mentioned four types.

The *Strategic* social action type is non-applicable in this study. This is due to the fact that within OSSD, innovations are legitimized by conducting rational deliberations amongst contributors of a certain project. More specifically, the majority of issues discussed within the Drupal open-

source project are concerned with solving bugs or implementing new features. To go forward with any suggestion a solid argument must be accepted by Drupal's founder, Dries, and the rest of the community. Since no member is under an obligation to accept, we assume in this study that the use of *strategic action* would be ineffective in this study. *Thus, strategic action* has not been identified from within the data collected.

Table 3-1 lists and explains the four social action types identified in ISS.

Figure 3-2 Explaining the Different Social Action Types

Social Action Type	Characteristic
Communicative Action	Oriented to achieve understanding. The intention of communicative actions is to make clear and comprehensible statements that are truthful and aims at establishing mutual understanding between the members of a conversation. This action is namely aimed at educating the recipient on something that the commenter is aware of. This social action excludes any sort of challenge within a discussion.
Discursive Action	Raises a challenge, whether it was in the form of challenging a validity claim presented by another person, or simply questioning another person's actions. In essence, any comment that raises some sort of challenge
Instrumental Action	Aims at directing or 'instrumenting' a discussion towards a certain direction. It is an action that attempts to exert influence within a certain social context.
Strategic Action	Is when it is not the speaker's sole purpose to achieve mutual understanding, rather it is a mean for a further end. The speaker could knowingly guide the hearer through particular inferences, for the purpose of achieving a goal that the speaker has, and that the hearer is unaware of.

3.3.2 Validity Claims

For the purpose of this study, Habermas's social action types are used to analyze the conversations that take place within the OSS community when participants propose, deliberate, and seek to legitimize new software innovations. From an analytical perspective, to critically analyze the legitimation process which unfolds during the rational deliberation one must identify the social action types involved and the specific validity claims that are raised and challenged (Ngwenyama & Lee, 1997).

In principle, each innovation proposal implicitly raises at least one of the validity claims below (Table 3-2) and other community members will only raise challenges to claims they find difficult to substantiate, thus only a subset of these might be challenged and debated upon in the rational deliberations. For example, if a participant proposes an innovation and another member does not

understand the proposal, then that member will be challenging the comprehensibility claim of that innovation. After which a conversation will be oriented to achieving an understanding of the nature of the proposal in an attempt to redeem the implicit comprehensibility claim challenged. Validity claims and challenges on them ensue in the dialogue. A deliberation might turn to some other validity claim, such as effectiveness or relevance, and the same process can take place. An innovation can only be accepted, or a new feature added, when such deliberations continue until agreement is reached on the issue at hand.

Table 3-1

Validity Claim	Criteria for Ideal Communication	Potential Breakdown	Validity Test
Comprehensibility	What is communicated is legible (audible) and intelligible	misunderstanding	Is the communication sufficiently intelligible? Is the communication complete? Is the level of detail appropriate?
Truth	Is the proposition content of what is communicated factual or true?	misrepresentation	Is the communication honest? Is the communication reasonable? Is the content of the communication warranted?
Relevance	Is the proposal relevant to the problem at hand?	inappropriateness	Does the proposed innovation offer a solution that directly addresses the defined problem?
Effectiveness	Is the proposal effective for resolving the problem?	ineffectiveness	Is the proposed innovation an effective solution for the defined problem? Are there better solutions for the defined problem?
Efficiency	Is the proposal efficient for resolving the problem?	Waste of resources	Does the proposed innovation make efficient use of the available resources? Are there more efficient alternatives?

3.3.3 Influence Tactics

In his elaboration of the Ideal Speech Situation, Habermas acknowledges that participants in rational deliberations exercise influence tactics to persuade one another. Depending on the social and organizational context, and the position of the social actor within the context, influence processes can differ between soft and hard tactics. In this study I am interested in the type of influence tactics the participants use during the debate process. For this part of the empirical study I have chosen influence tactics of Yukl and Falbe (1990), illustrated in figure 3.2 below.

Table 3-3 A List of Influence Tactics

Influence tactic	Definition
Rational persuasion	The agent uses logical arguments and factual evidence to persuade the target that a proposal or request is viable and likely to result in the attainment of task objectives.
Consultation	The agent seeks a target's participation in planning a strategy, activity or change for which the target's support and assistance are desired, or is willing to modify a proposal to deal with the target's concerns and suggestions.
Ingratiation	The agent uses praise, flattery, and friendly or helpful behaviour to get the target in a good mood or to think favourably of him or her when asking for something.
Personal appeals	The agent appeals to the target's feelings of personal loyalty and friendship when asking for something.
Exchange/reciprocity	The agent offers an exchange of favours, indicates willingness to reciprocate at a later time, or promises a share of the benefits if the target helps accomplish a task.
Alliance/coalition	The agent seeks the aid of others to persuade the target to do something, or uses the support of others as a reason for the target to agree as well.
Coercion/pressure	The agent uses demands, threats, frequent checking or persistent reminders to influence the target to do what he or she wants.
Rewards/recognition	The agent uses incentives and rewards to influence the target to achieve the task objectives.

(Yukl & Falbe, 1990)

4. Organizational Context of the Case

4.1 Background

Drupal was founded in 1999 and it all started when Dries Buytaert, the founder of the Drupal project, shared an internet connection with eight of his fellow students in the dorms of the University of Antwerp because it was too expensive for each to have their own connection. Since they were sharing the same connection, Dries felt that there was potential for an efficient medium of communication between them with the help of the central connectivity, so he started a simple internal site where they discussed news and any other items of interest to them, it was a blog for the eight of them. After he graduated, Dries hosted the website on the internet so that he and his friends can still use it, and allowed other people to become a part of it. Eventually, the website grew to become more than just a simple way for the eight friends to communicate. Dries, his friends, and the new audience from the internet started to discuss “new web technologies, such as moderation, syndication, rating, and distributed authentication. Drop.org slowly turned into a personal experimentation environment, driven by the discussions and flow of ideas. The discussions about these web technologies were tried out on drop.org itself as new additions to the software running the site” (Drupal, 2013). Within a year, the website received interest from developers around the world, and in an attempt to further the potential of the experimental software they were building, Dries made the project open-source in 2001 (Drupal, 2013).

Operations in Drupal are conducted by a volunteer community of over 630,000 users and developers. Operations are conducted on dedicated IRC channels, online forums, and during face-to-face conferences. Drupal.org, the drupal projects official website, state that contributors enroll from over 200 countries (in 181 languages) and thus the majority of discussions regarding bug reports, new feature requests and tasks take place online, either in IRC chats or online forums in Drupal.org. Conferences also frequently take place in counties all over the world, where seminars and opportunities to discuss various issues take place. “[the] worldwide community drives the innovation that makes Drupal the preferred choice for web developers and site owners.” (Drupal, 2013).

As new and innovative ideas kept coming into Drupal from contributors, innovative projects took flight and Drupal became a successful open-source content management system that was being used by individuals as well as institutions, some of which are key players in their industries, such as The Economist news publications, universities such as MIT and Harvard and corporate websites for the likes of AOL, MTV, SONY, and Warner Brothers Records (Drupal, 2013).

4.2 Design and Technical Information

Drupal is an open source content management software that provides tools to help build, organize, administer, and customize websites. The drupal software is designed in a way that allows individuals to build and customize websites without needing to write code. This is done by downloading the Drupal core software that runs the wide range of available modules that have already been designed and submitted by the community, for the use of anyone who chooses to use. Drupal is also able to cater to the more technical audience that require further customization. In such cases, developers can either modify existing modules, or write modules that are compatible with the core software. This is possible because the Drupal core is open source, thus the community can code modules that are compatible with the Drupal core. Also, developers and expert users can find support when developing new modules from the Drupal community. Essentially, the Drupal core can be downloaded to run the basic software, and customization of the software is done in the form downloading modules on a need basis.

The Technical Background of the Software

The Drupal core software consists of code that allows basic functionality such as a library of common functions, and basic modules. Modules add certain required functionalities such as Calendars, Image Galleries, or Forums (cf. Appendix A). These modules are added to the core on a need basis, and are called on by the core software with what Drupal names Hooks. Hooks are the internal events that call certain modules to “hook into” the rest of the Drupal software (VanDyk & Westgate, 2007).

In Drupal, Themes are responsible for translating the code into languages that the internet browsers recognize, such as HTML. This can be done via several of the most popular templating approaches such as the Template Attribute Language for PHP and PHP template.

In regards to strategy, Drupal's goal is to be able to run on inexpensive web hosting accounts such as Apache, which is an open source web server, and to be also able to distribute to the largest amount of websites. "The former goal means using the most popular technology, and the latter means careful, tight coding." (VanDyk & Westgate, 2007). To this end, Drupal is written in the PHP, which is a popular coding language that runs on all popular platforms.

4.3 Social and Organizational Structure

Essentially, Dries is the authority that must approve new innovations into the software core. In Drupal, the transition of an innovation from a mere suggestion into a legitimate item to be included in the next version of the software is called *Committing*. Only the **core committers** (Table 4-1) below are allowed to commit to Drupal. During the timeframe in which this study examined Drupal, the only core committer was Dries. By giving himself this authority, Dries is able to ensure the quality of new features to the software, and maintain the project on a roadmap towards strategies he sees fit.

In addition to the core committers, **Maintainers** are also members in the community whom have authority and a more significant opinion than others in the forums, based on past contributions to the Drupal project. Maintainers are appointed by Dries, and operate to take some load off Dries by being responsible for a certain area/module of the software. These maintainers have the authority to commit into their own modules only, whereas Dries has the authority to commit anywhere in the software. During the production of Drupal 4.6, Dries had assigned 10 site maintainers to different areas of the software such as the filter system, locale system, menu system, Blog API and so on. A complete list of the site maintainer and their responsibilities for 4.6 can be found here <https://api.drupal.org/api/drupal/MAINTAINERS.txt/4.6>.

Finally, core contributors may partake in an existing discussion that they can add value to, or create a new discussion in which they propose a new patch, feature, or discuss any issue relevant to the status of the Drupal project (cf. table 4-1). These contributions take the form of messages in an online forum or IRC Chat. A deliberation on the initiative takes place, and if progressed enough, an idea could be legitimated into the software, or debunked all together.

One can join the Drupal community by registering in www.Drupal.org. From there, all open issues can be viewed in the issues queue. The Drupal software project entails the below roles and responsibilities:

Table 4-1 The Different Roles in the Drupal Project

Drupal OSS Collaborators	Position and Role	Responsibilities
<i>Founder and Lead Developer</i>	Obligatory Passage Point Permanent Core Committer	approving or rejecting software innovation proposals and patches Appointing code maintainers
Any software developers who are members of Drupal OSS community	<i>Core committers</i> <i>Appointed by Dries for their extensive experience with and extensive knowledge of the Drupal software</i>	Reviewing software innovations proposals Maintaining software code Implementing software changes
	<i>Branch Maintainer</i> <i>Appointed by Dries ...</i>	The branch maintainer are allowed to commit code only in their respective branches
	<i>Maintainer</i> <i>Anyone may apply for maintainer status after they have made significant contributions to a specific software component</i>	Maintenance of a specified portion of the software (for example, a particular core module)
	<i>Core contributor</i> <i>Anyone who have made substantive contributions to a core component of the software</i>	Voluntary contribute software innovations, patches or documentation for the Drupal software

(Drupal, 2013)

5. Research Methodology

This research uses a critical theory methodology for interrogating the dynamics of organizational discourse (Ngwenyama & Lee , 1997; Cukier, Ngwenyama, Bauer, & Middleton, 2009). The critical theory approach to Information Systems (IS) research has a long history starting in the early 1980's (Lyytinen K. , 1992; Myers & Klein, 2011). The critical theory approach to IS research uses qualitative and quantitative methods and a critical interpretive approach to develop understandings about social actions within organizational situations (Lyytinen K. , 1992; Myers & Klein, 2011). This study uses the Critical Discourse Analysis (CDA) method developed by Cukier et. al (2009) as the primary strategy for analyzing the rational deliberations of the Drupal development team. The CDA method offers a strategy and a set of procedures for interrogating discourses to identify empirical observations concerning validity claims which are embedded in conversations. In this study I am interested in how the Drupal collaborators, engaged in rational deliberations, challenge and defend the set of validity claims raised by specific innovation proposals. The CDA uses Habermas's Communicative Action Theory (CAT) and content analysis for identifying empirical observations about validity claims. The empirical materials for this study comprises a corpus of 6,000 pages (in .doc format) of textual data documenting conversations of the Drupal open source development community between 2003 and 2004. The majority of these conversations discuss the builds of Drupal 4.5 and 4.6. While these conversations document the general interactions of the Drupal developers, they include the rational deliberations of the Drupal developers while engaged in decision making on software innovation proposals from members of the community. As such these conversations are a source of primary data on Drupal's software innovation legitimization processes, which is the empirical situation of interest to this thesis research.

5.1 Empirical Analysis Procedure

The empirical analysis of the research followed a multi-stage process in which the concepts of the theoretical framework were used to code, categorize and analyze the contents of conversations pertaining to the rational deliberations about innovation proposals in order to identify relevant empirical observations for theoretical analysis and interpretation. **Step 1** involved data sampling from the corpus of the empirical materials to identify deliberative

conversations concerning software innovation proposals relevant to this research and the preparation and loading of these textual data into HyperResearch for coding and analysis. **Step 2** involved three levels of analysis and coding. However, before coding the conversation threads I read and categorized them as Committed and Uncommitted innovation proposals. I then commenced the three rounds of coding of the selected conversations using the concepts of Communicative Action Theory (CAT), Influence Tactics (IT) and Validity Claims (VC). This process consisted of coding first the social action types, then the influence tactics and finally the validity claims. The validation process that followed each round of coding consisted of a set of rational deliberations between me and my supervisor. In these deliberations my supervisor would challenge a sample of my codes and I had to defend my coding decisions with regards to the concepts of the theoretical framework, sometimes this lead to the revising of code. This process was used to ensure that I produce defensible and valid interpretations of the empirical materials. **Step 3** involved identifying (a) different types of innovation proposals and their characteristics and (b) identifying and characterizing different categories of legitimization processes. While I was able to do in-situ coding to identify types of innovation proposals; I needed to analyze the legitimization processes in relation to the empirical observations of social action types, influence tactics and validity claims. In this way I was able to develop an empirically grounded understanding of the characteristics and dynamics of software innovations legitimization processes. I will now provide a brief summary of the empirical analysis before moving on to an in-depth discussion of the findings in the following two chapters (Chapters 6 and 7).

5.2 Summary of Empirical Analysis

Step 1: The empirical materials was received in the form of over 30,000 pages of textual data (MS Word format) harvested from the electronic communications of the Drupal community for the period 2004-2005. I started this empirical analysis with the idea to develop a sample of 100 complete conversations concerning deliberations on software innovation proposals. I started by reading empirical materials for the purpose of selecting appropriate conversations and planning the coding process. However, I soon realized that the average size of the conversations was quite large and that a strategy was needed to manage the process. So I started the data preparation process by printing the first 3000 pages of the empirical materials, reading the pages, looking for

clues that would enable me to identify the beginning, ending and content of a conversation thread. I found clues which I used to code the beginning and end points of conversation threads. Using this approach I was able to identify 78 conversations in the first 3000 pages sampled. However, on further more in-depth analysis of these conversations, three challenges arose:

1. Since the data was from email communications, conversation threads are repeated again and again with every new comment, resulting in long duplicated text.
2. Some discussions were too short to be able to act as evidence for anything
3. After the 78 conversation threads were scrubbed and prepared for empirical and coding there were no more than 12 complete conversations about unique software innovation proposals that were approved by the Dries and the community.




A total of 36 conversation threads were deemed invalid and removed from the dataset based on the above three constraints. This left me with 42 valid threads, only 12 of which constituted commits. I continued the data preparation process by selecting the next 3000 page segment of the textual data and repeating the process of preliminary analysis. However, this time using the MS Word Search feature I searched for keywords such as ‘Committed’ to help identify relevant conversations in an attempt to increase the count of conversations that constitute commits. In my re-reading of the text in the first segment of my preliminary analysis I realized that the word ‘committed’ was explicitly stated every time an innovation proposal was accepted for implementation. I was able to find 12 of this valid type of conversations, which were added to my dataset. At this point, the empirical data comprised a set of 24 committed conversations and 30 uncommitted conversations for a total of 54 valid conversations comprising 6,000 pages of textual data. This was too large of a file to load in HyperResearch, it slowed the analysis significantly. So it was necessary to re-load the data set into HyperResearch as 9 smaller files to speed up machine processing for coding and content analysis.

After the text was loaded into HyperResearch, two empty cases were found in the uncommitted category, both named ‘unnamed’. I had lost these files, to correct for this loss; those two uncommitted cases were eliminated from the count of the dataset. The dataset’s final count consisted of 52 deliberative discourse cases, 24 of which consist of successful legitimations (A.K.A commits in Drupal), while the other 28 deliberations did not result in a commit, at least until the point of the deliberation that was examined in this study.

Step 2a: After loading the textual data into HyperResearch I commenced the coding process. I then commenced coding each conversation using the concepts from the CAT. The coding procedure consisted of reading and re-reading the 52 conversation threads to develop an understanding of the context and dynamics of the conversation. After a significant amount of reading and re-reading I started coding occurrences of the primary social action types (communicative, strategic and instrumental) of each comment in each thread based on definitions I had developed in my codebook. Thus, three codes were created in HyperResearch (Communicative Action, Discursive Action and Instrumental Action).




In the 24 committed conversations there are a total of 221 comments. As illustrated in figure 5-1, evidence shows that communicative action is the most common social action type engaged by members in these cases. Usually, but not always, Dries is the person in these deliberations to use instrumental action; often in the form of giving directions to an innovator to accomplish a commit (cf. Appendix B). Figure 5-1 summarized each social action type in these 24 committed cases.

Figure 5-1 Summary of social action types for 24 committed cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
CAT - Communicative	187	2	39	7.792	7.396	
CAT - Discursive	26	0	6	1.083	1.472	
CAT - Instrumental	8	0	4	0.333	0.868	

In the 28 uncommitted conversations there are a total of 430 comments. As illustrated in figure 5-2, evidence shows that communicative action is again the most common social action type engaged by members in these 28 cases. As with committed, in uncommitted cases Dries is usually, but not always, the person to use instrumental action; often in the form of giving a set of criteria that must be met before Dries considers a commit. Figure 5-2 summarized each social action type in these 28 uncommitted cases.



Figure 5-2 Summary of social action types for 28 uncommitted cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
CAT - Communicative	339	2	38	12.107	10.068	
CAT - Discursive	83	0	8	2.964	2.589	
CAT - Instrumental	8	0	2	0.286	0.659	

Step 2b: The second coding activity consisted of coding for influence tactics within each conversation thread. I read each comment within the context of the conversation flow and labeled according to my understanding of its meaning. Using the influence tactics aspect of the framework I analyzed each comment to determine which of the influence tactics were enacted in it and a corresponding code was assigned. The influence tactics applicable for the context of the Drupal discussions were either the Rational Persuasion or Consultations tactic.





In the 24 committed cases, there are 85 instances of rational persuasion tactics and 15 instances of consultation tactics. Figure 5-3 provides a summary of the empirical observations made concerning influence tactics used in the 24 committed cases.

Figure 5-3 Summary of Influence Tactics in 24 Committed Cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
IT - Consultation	15	0	4	0.625	1.209	
IT - Rational Persuasion	84	1	18	3.5	3.612	

In the 28 uncommitted cases, there are 211 instances of rational persuasion tactics, 19 instances of consultation, 2 support/alliance tactics and 1 instance of ingratiation. Figure 5-4 provides a summary of the empirical observations made concerning influence tactics used in the 28 uncommitted cases

Figure 5-4 Summary of Influence Tactics in 28 Uncommitted Cases





Code	Total	Min	Max	Mean	Std Dev	Bar Graph
IT - Consultation	19	0	3	0.679	0.945	
IT - Ingratiation	1	0	1	0.036	0.189	
IT - Rational Persuasion	211	1	27	7.536	6.438	
IT - Support/Alliance	2	0	1	0.071	0.262	

In both cycles of coding each coded text segments were marked for future retrieval and memos were written to document my rational for the interpretations. These memos were then indexed to

the specific text segments using the HyperResearch indexing and memo tools. After completing each cycle of coding I took time out from the empirical analysis and focused on reading relevant literature and re-working parts of the early chapters of my thesis. This break from the empirical analysis enabled me to re-examine the coding a second and third time with a more critical eye, sometimes revising my coding. I then had discussions with my thesis advisor to clear points of concern I had about any of the interpretations I was making. His approach was to challenge me to defend my position or to modify it in the light of our discussions. It is important here that it took time and intense reading for me to develop an understanding of the social and organizational context of Drupal. And as I progressed in my understanding, I had to revise some of my coding.





Step 2c: In this step of the empirical analysis I focused on identifying which validity claims were challenged and redeemed within the 52 cases. Using concepts identified in Table 3.2 I had to closely examine the dynamics of the conversations to explicitly identify challenges made to innovation proposals and coded them in accordance with the five types of validity claims outlined in my framework. I was also careful to code the replies to challenges and to document the discourse cycle from beginning until the challenge was redeemed or it prevailed. In the 24 committed cases, efficiency claims were challenged most frequently, followed by effectiveness, comprehensibility then relevance. Figure 5-5 below summarizes the primary observations on the validity claims challenged and resolved for the 24 committed conversation threads.

Figure 5-5 Summary of the Validity Claims Challenged in 24 Committed Cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
Comprehensibility	3	0	1	0.125	0.338	
Effectiveness	6	0	2	0.25	0.532	
Efficiency	18	0	4	0.75	1.113	
Relevance	2	0	1	0.083	0.282	




In the 28 uncommitted cases the frequency of validity claims challenged is in the same sequence as the 24 committed cases. Efficiency claims were also challenged most frequently, followed by effectiveness, comprehensibility then relevance. The Figure 5-6 below illustrates the validity claims challenged for the 28 uncommitted cases.

Figure 5-6 Summary of Validity Claims Challenged for 28 Uncommitted Cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
Comprehensibility	13	0	3	0.464	0.881	
Effectiveness	16	0	3	0.571	0.959	
Efficiency	43	0	6	1.536	1.815	
Relevance	11	0	2	0.393	0.685	

Step 3: At this stage of the empirical analysis I switched focused to identifying specific categories of innovation proposals and characterizing the legitimation processes on the basis of examining the empirical observations made in the prior coding exercise. *Step 3a:* On closer reading of the conversations I was able to determine that were three types of innovation proposals: Bug Fixes, New Features and New Tasks. I then did content analysis to find out which conversations were discussing which type of innovation proposals (Bug Fixes, New Features, or New Tasks) and coded each conversation accordingly. Each conversation discussed only one of those three types of innovations. For the 24 committed cases, the analysis yielded 11 empirical observations of bug fixes, 6 of new features, and 7 of new tasks. Figure 5-7 below summarizes the types of innovations for the 24 committed cases.

Figure 5-7 Summary of Types of Innovation Proposals for 24 Committed Cases

Code	Total	Min	Max	Mean	Std Dev	Bar Graph
Bug Report	11	0	1	0.458	0.509	
Feature Request	6	0	1	0.25	0.442	
Task	7	0	1	0.292	0.464	

For the 28 uncommitted cases, those frequencies are 13 feature requests, 12 bug fixes and 3 tasks. Figure 5-8 below shows the types of innovations for the 28 uncommitted cases

Figure 5-8 Summary of Types of Innovation Proposals for 28 Uncommitted Cases




Code	Total	Min	Max	Mean	Std Dev	Bar Graph
Bug Report	12	0	1	0.429	0.504	
Feature Request	13	0	1	0.464	0.508	
Task	3	0	1	0.107	0.315	

Table 5-9 below provides three examples of empirical observations of how these different types of innovation proposals are identified in the conversation threads. As observed in the content analysis, bug fixes are proposals aimed at addressing bug reports in the software. In such

deliberations, a member of the community starts the conversation by identifying a specific bug, and the rest of the community joins the deliberation. For example, in thread 57 in Figure 5-9, Jasper, a member of the Drupal community identifies a bug that needs fixing and explains the problem. While in thread 75, we see kbahey is proposing a new software feature to enhance the Drupal software. Finally in thread 74 Drumm is seeking to update a software module's instructions text, to remove what he thinks are confusing instructions.

Figure 5-9 Examples of Different Types of Innovations

Innovation Type	Case	Empirical Evidence (Quote)	Explanation
Bug Fixes	Thread 57	<p>November 19, 2004 - 16:03 : jasper</p> <p>1. "my account" link can not be given "weight"</p> <p>2. If edited, "my account" link behaves strangely: appears as separate menu in admin screen, disappears from navigation menu, etc.</p> <p>3. Subitems can't be added under my account, or, strange things happen if you try to do that.</p>	Jasper identifies a bug in the software
New Features	Thread 75	<p>February 1, 2005 - 21:26 : kbahey</p> <p>Attachment: http://drupal.org/files/issues/contact.module-subject.patch (1.93 KB)</p> <p>I find it very undescriptive when I recieve a message from Drupal with the subject "message from username".</p> <p>This patch adds a "subject" field for the contact.module which the user can fill, and would tell you what they want right away. Oh, and it helps group the 'conversation' on Gmail into something meaningful.</p> <p>(Note, I have not tested this since I do not have a CVS installation at the moment. Appreciate if someone can test it).</p>	This feature will allow more efficient communication within the Drupal community.
New Tasks	Thread 74	<p>February 8, 2005 - 01:10 : drumm</p> <p>Attachment: http://drupal.org/files/issues/page.module_2.diff (1.56 KB)</p> <p>The page module's long help text is a bunch of lies and then it briefly explains it's permissions. IMO it should just be taken out. I can't think of what help should be there.</p>	Drumm is seeking to eliminate excess wording in Drupal.

In all cases when an innovation proposal is made the members engage in open deliberations at the end of which, the proposal is either accepted or the conversation is left open until updated in a repeat attempt at approving the innovation discussed. In these deliberations the members of the Drupal community focus on clarifying the problem, designing and implementing a solution that improves the functionality, reliability, effectiveness and efficiency for both the developers and end-users.

Step 3b: I focused on analyzing the legitimation processes with an interest in identifying similarities and differences among the conversation threads based on the profile of social action types, influence tactics employed and the frequency of challenged to validity claims. Based on the aforementioned criteria, I was able to find two distinct categories of legitimation processes summarized in Figure 5-10.

I came about distinguishing these two categories during the content analysis of the committed cases, that is when I noticed that some conversation were taking much more collaborative effort to arrive at a commit than others. I then created a code for short commits and another for long commits, and assigned each case with the appropriate code, and then I summarized the characteristics of all cases in the short commit category based on the three aspects of the framework this study uses, and again for all the long commits. This was done in HyperResearch. I found that the characteristics of each category were different: (1) the frequencies of the types of innovation (whether bug fix, new feature or task), (2) complexity of innovation (high vs. low) and (3) frequencies of types of validity claims challenged. This result was in line with my conjecture that there must be some aspects of the cases that influence the length or amount of collaborative effort of conversations. These two different innovation legitimation processes will be elaborated upon in Chapter 6.

Figure 5-10 Summary of Two Categories of Legitimation Processes

<u>Characteristics</u>		Fast Commits	Slow Commits
Number of cases		16	8
Complexity	Low	11	1
	High	5	7
Social Action Types	<i>Communicative</i>	75	112
	<i>Discursive</i>	8	18
	<i>Instrumental</i>	2	6
Influence Tactics	<i>Rational Persuasion</i>	32	52
	<i>Consultation</i>	6	9
Validity Claims Challenged	<i>Comprehensibility</i>	2	1
	<i>Effectiveness</i>	3	3
	<i>Efficiency</i>	5	12
	<i>Relevance</i>	1	1

6. Discussion of Empirical Findings

As stated earlier there are 52 empirical cases of innovation proposals analyzed in this study. Of these 28 were uncommitted and remain unapproved at the end of my study, while 24 were approved, committed and implemented in the Drupal core software. The primary objective of this chapter is to outline some of the key dynamics of the legitimation processes of the committed software innovations. However, for the sake of the balance of the empirical observation of this thesis I need to briefly discuss the key characteristics of the uncommitted innovation proposals before moving on to discussing the key dynamics of the legitimation processes of committed cases. The legitimation processes for both types (committed and uncommitted) unfolded as a structured rational deliberation (SRD) which comprises a set of communicative interactions that can be summarized as follows:

1. A community member announces and describes a proposed innovation or describes the need for an innovation and consults with the community about a solution
2. When an innovation proposal is offered other Drupal community members will respond, accepting, challenging or rejecting the validity claims of the proposed innovation, or proposing alternatives to the one under discussion
3. The proponent may seek to defend the validity claims of the proposed innovation, modify it in response to community input or abandon it all together in favor of another alternative
4. When a community member describes the need for an innovation and consults with the community about a solution, the other members will respond presenting alternative approaches to ones already presented
5. At some point in the deliberations Dries will intervene accepting the proposed innovation for inclusion in the core software or request further deliberations outlining some outstanding legitimation issues
6. When Dries accepts the innovation proposal, he states publicly ‘committed’ and makes the innovation proposer responsible for finalizing the software code, its testing and inclusion into the Drupal core modules

From the perspective of Dries the objective of these SRDs was to obtain community involvement in vetting the innovation proposals in order to achieve high quality software and to identify a community member who would take the responsibility for implementing (and possibly

maintaining) the software innovation. From the perspective of the innovation proposer, the deliberations served as a forum to obtain input and support from the community and to achieve approval and recognition for the proposed innovation from Dries. In order to achieve success the proponent had to convince a critical mass of the community represented by an appropriate number of ‘+’ votes on the value of the innovation proposal (more details to follow).

6.1 Basic Characteristics of Uncommitted Innovation Proposals

The empirical analysis of the 28 uncommitted innovation proposals revealed 14 high complexity and 14 low complexity proposals. Of those, 12 are Bug Fixes, 13 are New Features and 3 are New Tasks. Empirical analysis of the conversations also showed that a range of 1 to 12 and an average of 5 actors participated in the SRDs of these uncommitted innovation proposals. The SRD cycles of these uncommitted innovation proposals comprised a total of 430 comments and an average of 15 communications per deliberation. These communications comprise a range of 2 to 38 communicative actions, 0 to 8 discursive actions and 0 to 2 instrumental actions. Table 6.1 below summarizes the basic characteristics of the 28 uncommitted cases. I will discuss a few these in more detail later when I discuss the deliberation processes.

Figure 6-1 Basic Characteristics of 28 Uncommitted Innovation Proposals

Empirical Dimensions	Uncommitted Cases	
Number of Cases	28	
Complexity of Innovation	High	Low
	14	14
Type of Innovation	Bug Fixes	12
	New Feature	13
	New Task	3
Number of Actors in Deliberations	Range	Average
	1-12	5
Number of Comments in Deliberations	Total	Average
	430	15

6.1.1 Communicative Characteristics of SRDs

The structured rational deliberations (SRDs) unfolded as a set of dialogues in which the actors used various forms of communicative actions to influence each other towards the development of consensus about the value of an innovation proposal to the Drupal core software. The SRD cycles concerning the uncommitted innovation proposals vary in length, but it is important to make clear that at the time of this study these conversations were still in progress. Therefore it is not possible to state whether they were eventually legitimized. However, it is possible to discuss the communicative characteristics of these SRDs. My empirical analysis of these SRDs suggests the participants were oriented to reaching agreement and had a preference communicative action and soft influence tactics. The use of communicative action was 3.75 times more than all other action types. There were 83 empirical observations of discursive action and 8 of instrumental action. The dominant influence tactic was Rational Persuasion which was enacted 11.2 times more than all other influence tactics. However, there were 19 empirical observations of the consultation influence tactic. There were 83 validity claim challenges, most, 43 focused on the efficiency of innovation proposals, while 16 focused on their effectiveness, 13 on comprehensibility and 11 on relevance. Figure 6.2 below summarizes the key communicative characteristics of 28 uncommitted cases. To provide a better understanding of the communicative dynamics of the uncommitted innovation proposals I will discuss an example in the next section.

Figure 6-2 Communicative Characteristics of the Uncommitted Innovation Proposals

ISS Dimensions	Uncommitted (28 Cases)			
<u>Action Types</u>	Total	Min	Max	Mean
Communicative	339	13	38	12.11
Discursive	83	7	8	2.96
Instrumental	8	1	2	0.29
<u>Influence Tactics</u>	Total	Min	Max	Mean
Rational Persuasion	213	1	27	7.54
Consultation	19	1	3	0.68
<u>Validity Claims Challenged</u>	Total	Min	Max	Mean
Comprehensibility	13	0	3	0.46
Effectiveness	16	0	3	0.57
Efficiency	43	1	6	1.54
Relevance	11	0	2	0.39

6.1.2 Examples of SRDs of Uncommitted Innovation Proposals

These two examples of uncommitted innovation proposal (Cases 2 and 6) represent the extreme points of uncommitted cases empirically analysed in my study. Because these cases have not been rejected there is no way of stating definitively their statuses. However, what I can do is to give empirical observations of the communicative characteristics based on my analysis of the continuing deliberations on these cases. Case 2 is a New Feature of high complexity while Case 6 is a Bug Fix of low complexity. The SRD for Case 2 comprised 8 participants involved in 47 communications comprising 28 instances of influence tactics and 6 validity claim challenges. The SRD for Case 6 comprised 5 participants involved in 20 communications comprising 11 instances of influence tactics and 7 validity claim challenges. Figure 6.3 provides a summary of the details of the communicative characteristics of the two uncommitted cases. In the following section I will provide short selective illustrations of the SRD communications on Case 2 to provide some understanding of the communicative dynamics.

Figure 6-3 Communicative Characteristics of Two Uncommitted Examples

ISS Dimensions	Case 2(Uncommitted)	Case 6(Uncommitted)
Action Types	Total	Total
Communicative	38	13
Discursive	7	7
Instrumental	2	1
Influence Tactics (Total)	28	11
Rational Persuasion	27	9
Consultation	1	2
Validity Claims Challenged (Total)	6	7
Comprehensibility	3	3
Effectiveness	2	0
Efficiency	1	4
Relevance	0	0

6.1.3 Selective Illustration of the SRD Dynamics of Case 2

Case 2 concerns a New Feature of high complexity proposed by Chx. Chx started the deliberations and replied 25 times throughout the conversation exerting rational persuasion influence tactics. Dries contributes to the conversation a total of 11 times, while Jose A Reyero contributes to the conversation using two instances rational persuasion and one instance of consultation. The rest of the participants contribute with a total of 8 direct replies to rational persuasion tactics in both a communicative and discursive manner, and in 2 instances full support was given to the proposed innovation. None the less, it remained uncommitted at the end of the period of conversations included in this empirical study.

The SRD is initiated by Chx who has developed some new functionality and is suggesting that it should be considered for inclusion in Drupal core software. Chx states

(December 23, 2004 - 23:25):

"Attachment: http://drupal.org/files/issues/node_builder.patch (5.56 KB)

Although I have posted this to the devel list, as Steve pointed out, putting patches to sandbox is not the best way to do things. So I open this thread. I have measured the time for build a node query in this is about 0.22-0.24ms (on an Athlon 933 MHz machine)."

A few hours later Dries responded making it clear that he is unlikely to commit the innovation proposal. He used discursive action, challenging the effectiveness on the innovation proposal:

December 24, 2004 - 00:33 : Dries

I'm not likely to commit this. It's not conform with Drupal's coding conventions, but more importantly, a node query builder doesn't solve any real problems. It just adds a different way of doing things without offering a significant advantage.

Almost immediately Chx responds using rational persuasion to make a case for the effectiveness of the innovation, pointing to some specific functions that could find it relevant. He also retests his proposed software innovation and discovers some problems, reports on them and points out that it is just the beginning of the SRD, so the problems can be worked out with revisions:

December 24, 2004 - 00:51 : chx

Attachment: http://drupal.org/files/issues/node_builder_0.patch (5.67 KB)

*At this moment there are direct queries into the node table everywhere. After node_access_*_sql calls were invented, a lot of queries needed to be changed, and I think we will find more to be inserted. If someone does sg. else with the queries, permission, language etc. he needs to patch 60+ queries in core alone. This is not a good thing. That's why I am proposing a central node query builder. As for code standards, code-style.pl node.module returns 17 errors, none of them comes from my patch. However, I should admit, there were a few spaces missing from the database.inc patch. So I resubmit.*

December 24, 2004 - 00:56 : chx

Attachment: http://drupal.org/files/issues/node_builder_1.patch (5.67 KB)

OK, I was wrong, there is a space missing in one of the rewritten queries. But I doubt this matters too much, as this is only a proposal and I think there will be a lot of revisions before it gets to the core.

At this point Drumm enters the SRD replying to Chx's rationale and also challenging the effectiveness of the innovation proposal, but suggests some revisions, and lays out some conditions under which he would give support:

"December 24, 2004 - 01:26 : drumm

The blank lines between the comments and function starts will confuse the documentation parser on drupaldocs.org.

I'm not sure if I support this quite yet. Reducing the number of queries and whatnot is great, but this patch doesn't do that quite yet. I'd only want this if the new system is more readable and easy to write. I'm not sure if this is true.

As for code-style.pl, I don't think this is actually used much and it doesn't catch everything. Such as, most operators (=, >=, +, etc) like a space on either side of themselves. I'd support removing code-style.pl and throwing it in contributions

somewhere. (If this needs further discussion or action, please file another issue, lets keep this focused on this patch).

I'm going to wait for a few more revisions of this before I weigh in a

+1 or -1."

A few minutes later Jose entered the SRD in response to Chx's rational persuasion tactic, using communicative action and in support of Chx's innovation proposal:

December 24, 2004 - 01:58 : Jose A Reyero

I can really think of a number of advantages of this, and though not really for committing it yet, think is very good to see some discussion on the issue and some concrete implementations.

When implementing i18n I run into this problem, of having to patch lots of queries - actually too many- in about all the modules doing any kind of node listing. This far, I'm stuck at this point. Then I realized it was just the same for node permissions, and will be the same again for whatever new functionality you want to add in the future which affects node listings.

I think actually that direct access to 'node' table should be avoided when possible out of the node.module. And the only way to achieve this is to have some kind of query builder. And also think that chx's approach is very good. it provides some upper layer at the 'node'

level, which can use some object level semantics while relying on a db layer query builder, which could be used later for other objects -users, taxonomy terms...-

Just think of all the work that this could have saved when implementing permissions system, which I think is ugly and dirty anyway because of all that node_join, node_where...

In the long run, adding such complex logic as sql code, patching and re-patching queries is bad, very bad, making any new change one higher level of complexity and driving us away from a OO model, which I think we should tend to. It's not using PHP objects. It is thinking of nodes as 'objects' and coding according to it.

Oh, yes, sorry, this is growing too long to write it here :-). But we could talk also about db portability...

A fourth participant, Moshe, enters the SRD using communicative action, replying to Chx's rational persuasion in support of the effectiveness of the innovation proposal:

December 24, 2004 - 02:14 : moshe weitzman

I should add that the Organic Groups module could benefit from this node listing SQL builder. I have awkwardly worked around the problem for now but I don't like it.

I don't know if this patch is good or not, but I do lean toward the functionality that it offers. Remember that any developer is free not to use the query builder and issue direct SQL as needed. If the query builder is making life hard, simply don't use, just like today.

At this point a fifth participant, Kessels, enters the SRD using communicative action and in support of the relevance of the innovation proposal and votes '+1' in favor of its adoption:

December 24, 2004 - 11:35 : B?r Kessels

I +1 for this functionality (i have no time to comment on coding style etc).

It will not only improve maintainability, but will allow a far easier implementation of the i18n. i18n currently has a lot of patches, simply to add logic to all those node sql queries referred to.

Chx then presents a revised version of the proposal and uses rational persuasion in another attempt to obtain support from Dries and the other Drupal community members:

December 24, 2004 - 14:20 : chx

Attachment: http://drupal.org/files/issues/node_builder_2.patch (1.63 KB)

OK, maybe the former approach was too complex. How about this? This requires a lot less change to the queries and no change to database.inc and ten times faster.

As usual, a sample query is rewritten and an example of the proposed hook is provided.

Some hours later Chx follows up yet another revision of the proposal which he claims is more efficient than the last; here again he uses communicative action and rational persuasion.

December 25, 2004 - 01:09 : chx

Final thoughts for this day. If you like the last version, there is a possibility to automate the whole thing by adding the following three lines to the beginning of db_prefix_tables:

Of course, there is very small chance of a loop here, so this would require more thought, but I think at the end this would be a good thing.

Dries responds more positively, however, using communicative action and still seeking more input from other members of the Drupal community. He also politely requests that Chx does more query updates to make sure that the innovation is effective in all cases:

December 27, 2004 - 13:55 : Dries

I like the second approach better as it keep things readable and gives me a bit more control over the order in which things are written.

Looking for more feedback from the others. Did you try updating more queries to make sure it works as intended in all cases?

Chx complies with Dries's requests for testing, discovers some issues and revises the innovation proposal yet again. He responds to Dries as follows:

December 27, 2004 - 20:48 : chx

Attachment: http://drupal.org/files/issues/node_builder_3.patch (1.23 KB)

Yes, I tried with many different queries. And I have found one bug -- forgot the underscore in the regexp following FORM so I have rewritten the whole regexp. It is now a lot simpler :)

And node_query is now reentrant, so if a function in the process of hook_sql calls node_query, it won't fall into an infinite loop. This situation did not occur but it's better to forestall such things.

Moshe again enters the deliberations in support of the evolving innovation. He uses communicative action and is in support of the innovation, with some comments to spare for Chx's innovation proposal:

December 28, 2004 - 06:48 : moshe weitzman

This patch is a step forward. I thought of another feature which becomes easier with the proposed hook_sql(). think of filtering like freshmeat.net. At freshmeat, you can say that you only want to see projects where OS=Windows and License=GPL (for example). A Drupal equivalent is 'only show me nodes related to Democrats and Poverty'.

This sort of global node filtering is very hard in Drupal today. With this hook, it becomes easy.

Dries again responds with communicative action, although he is more supportive he offers some recommendations for Chx to do before a commit is possible:

December 28, 2004 - 13:37 : Dries

It's starting to look good (and you're getting more support)! Have you read the following threads:

<http://lists.drupal.org/archives/drupal-devel/2004-11/msg00198.html> and

<http://lists.drupal.org/archives/drupal-devel/2004-11/msg00230.html>?

It's an open issue related to your work. If you start joining tables, you might have to add DISTINCT(). When you are not joining tables, it would be nice if we'd not pay the cost of a DISTINCT().

It would be good if you could update the node queries in core to take advantage of it. It's not necessary while prototyping though.

It would be good if the il8n team could take a closer look at this patch by trying to use it. It would be good if you could add some PHPDoc.

Chx continue to repost updated version of the patch, but a little later in the conversation, Jose find a problem with the patch, but instead of solely challenging Chx's proposal, Jose attached a patch of his own:

January 7, 2005 - 18:30 : Jose A Reyero

Attachment: http://drupal.org/files/issues/central_node_query_node_module.patch (7.07 KB)

I found some problems when using the patch. The where conditions are not merged well if there are more than once, and also conditions need some parenthesis around.

So I replaced the \$where and \$join strings in _node_rewrite_sql by arrays which are imploded at the end.

Also added some 'hint' definitions. Patches for il8n module are coming next.

This works like charm for node listings, comment listings, searches, etc...

Btw, I also think 'semaphores' are not really needed.

This SRD continues with several more participants joining the deliberations, some supporting Chx's proposal, some challenging its validity claims, but I will stop the illustration here. The communicative dynamics of the SRD for Case 2 should be clear. Also it is clear that Chx is proposing a complex innovation and that the SRD is serving a critical function of improving the quality of the innovation. The roles of the other participants are also clear. In some cases they challenge validity claims of the innovation proposal in other cases they suggest alternatives to improve the efficiency and effectiveness of the proposal. However, as stated earlier this innovation was still uncommitted; the deliberations were still ongoing at the end of this study.

6.2 Basic Characteristics of Committed Innovation Proposals

In my empirical analysis of the 24 committed innovation proposals, I categorized 16 as Fast Commits and 8 as Slow Commits based on the number of communicative interactions required for them to be committed. Figure 6.4 below summarizes the basic characteristics of the 24 cases. My empirical analysis revealed that the 16 Fast Commits comprised 5 high complexity and 11 low complexity innovations, while the Slow Commits comprised 7 high complexity innovations and 1 low complexity innovation (cf. Appendix C, D). The Fast Commits comprise 6 Bug Fixes, 4 New Features and 6 New Task; while Slow Commits comprise 5 Bug Fixes, 2 New Features and 1 New Task. Empirical analysis of the conversations showed that 2 to 6 actors participated in the structured rational deliberations of the Fast Commits, while 4 to 9 actors participated in the Slow Commit deliberations. This resulted in significantly longer structured deliberations on Slow Commits. The SRD cycles of Slow Commits consist of 7 to 39 communicative action type interactions; while SRD cycles of Fast Commits consist of only 2 to 7 communicative action type interactions. I will discuss these in more detail later when I discuss the deliberation processes.

Figure 6-4 Summary of Basic Characteristics of the Case of Innovation Proposals

Empirical Dimensions	Fast Commits		Slow Commits	
Number of Cases	16		8	
Complexity of Innovation	High	Low	High	Low
	5	11	7	1
Type of Innovation	Bug Fixes	6	Bug Fixes	5
	New Feature	4	New Feature	2
	New Task	6	New Task	1
Number of Actors in Deliberations	Range	Average	Range	Average
	2 to 6	3.5	4 to 9	5.9
Number of Comments in Deliberations	Total	Average	Total	Average
	85	5.3	136	17

6.2.1 Communicative Characteristics of SRDs

The SRDs unfolded as a set of dialogues in which the actors used various forms of communicative action to influence each other towards the development of consensus about the value of an innovation proposal to the Drupal core software. The cycle of SRDs of Fast and Slow Commits vary in length as the Slow Commits required much more discussion than the Fast Commits. However, the type of communicative actions chosen by the participants in the deliberations suggests an orientation to reaching agreement and preference for soft influence tactics. The deliberations were dominated by communicative action and rational persuasion, with discursive action and consultation secondary options, and instrumental action rarely used. Validity challenges during the SRD cycles were mostly to the efficiency of the innovation proposal; and there were no meaningful differences in the type of validity claim challenges between Fast Commits and Slow Commits except for the number challenges (cf. Figure 6.5). Slow Commits receives more than 2 times the number of challenges in total. Each of the categories appears to arrive to the point of legitimization from different grounds, with the most obvious distinctions being that a Fast Commit occurs when the value of the fix seems to be of relatively clear or the bug fixes are of low-complexity, whereas Slow Commits are more complex and had many validity claim challenges, thus requiring more deliberative effort to arrive at legitimization. Table 6.5 presents a comparative analysis of the key empirically observed differences between Fast and Slow Commits. In the next section I will discuss some examples of SRDs of Fast and Slow Commits to illustrate the differences in their dynamics.

Figure 6-5 Comparison of the Communicative Characteristics of Fast and Slow Commits

ISS Dimensions	Fast Commits (16 Cases)				Slow Commits (8 Cases)			
<u>Action Types</u>	Total	Min	Max	Mean	Total	Min	Max	Mean
Communicative	75	2	7	4.7	112	7	39	10.4
Discursive	8	0	3	0.5	18	0	6	2.3
Instrumental	2	0	1	0.13	6	0	4	0.8
<u>Influence Tactics</u>	Total	Min	Max	Mean	Total	Min	Max	Mean
Rational Persuasion	32	1	5	2	52	2	18	6.5
Consultation	6	0	4	0.38	11	0	6	1.4
<u>Validity Claims Challenged</u>	Total	Min	Max	Mean	Total	Min	Max	Mean
Comprehensibility	2				1			
Effectiveness	3				3			
Efficiency	5				13			
Relevance	1				1			

6.2.2 Examples of SRDs of Fast Commits

I will now briefly discuss a few examples of Fast Commits to illustrate their communicative dynamics. The empirical observations used in these illustrations were derived from the coding and content analysis of the conversations during the SRDs. There are 16 cases of Fast Commits, with a total of 59 participants in the SRDs of them. That is an average of 3.7 participants per case, and a range between 2 to 6 participants per case. However, in 75% of the cases the SRDs consisted of the proponent and one other participant, after which Dries approved them for inclusion in the Drupal core software. The length of the SRD cycle is based on the number of communication interactions in it. The 16 cases in this category had a total of 85 communication interactions. Hence, there is an average of 5.3 communication interactions. Within the 16 cases, it is observed that there were 75 communicative action types, aimed at established mutual understanding, 8 discursive action types, which challenged validity claims, and 2 instrumental action types, which were mostly enacted by Dries. Rational Persuasion and Consultation were the primary influence tactics used in the deliberations of the 16 cases of Fast Commits. Rational Persuasion was used 5 times more than Consultation. There are 32 empirical observations of Rational Persuasion tactics and only 6 empirical observations of Consultation tactics in the

deliberations of the 16 cases. Of the 32 instances of the Rational Persuasion influence tactics observed in the deliberations of Fast Commits, 29 were enacted by the initial proposer of the innovation. Also, of the 6 instances of consultation tactics, 3 were enacted by the proposer as a strategy for initiating the SRD process.

6.2.3 Illustrative Examples

The three Fast Commits I want to discuss are Cases 84, 78 and 76 whose basic characteristics are summarized in Figure 6.6 below. Case 84 is a proposed fix in response to a bug report, and a low complexity type innovation. The Proposer of the Bug Fix is Uwe Herman, and there are four other participants in the deliberations, Morbus Iff, Stefan Nagtegaal, JonBob and Dries. In this SRD process Uwe states; “I ran ispell over the whole Drupal code (including themes etc.). Here's a patch with the fixes I (or ispell) found”. Morbus responds with a ‘+’ signaling support for the Bug Fix, while Stefan responds with an interrogation; “Did you also run iSpell over the helptexts?”. Uwe replies; “Which help texts? The ones embedded in any *.module oder *.inc file: yes. I scanned all plain-text files.” At this point JonBob enters the deliberations supporting the Bug Fix by stating; “I read through the patch, and all corrections appear to be... well... correct. +1.” At this point Dries legitimizes the proposal stating; “Committed to HEAD. Thanks!”, and assigns Uwe to implement the software update. Case 84 is legitimized after a short rational deliberation which comprised communicative actions, a single clarification, and no discursive challenges to validity claims (cf. Figure 6.7).

Figure 6-6 Summary of Characteristics of the Three Example Fast Commit Cases

Empirical Dimensions	Case 84	Case 78	Case 76
Type of Innovation	Bug Fix	Bug Fix	New Task
Complexity of Innovation	Low	Low	High
Number of Actors in Deliberations	5	3	5
Number of Communications in Deliberations	7	5	8
Number of Influence Tactics in Deliberations	1	2	1

Case 78 is also a proposed fix in response to a Bug Report. The Bug Fix proposed by Killes is a low complexity type innovation. There are two other participants in this SRD process, Uwe Herman and Dries. Killes starts the legitimation process by stating; “Drupal emits an annoying

number of PHP notices. The attached patch fixes a few of them in tablesort.inc.” Uwe responds to the proposal with corrections stating; “The patch looks broken, there's Email-Headers included (probably erroneously cut'n'pasted there?)”. Killes responds; “Oops, thanks.” Dries legitimizes the innovation proposal, stating; “Committed to HEAD”, and assigns Killes to implement the software update. Case 78 is legitimized after a short rational deliberation comprising two communicative actions, one discursive action challenging the effectiveness of the original proposal, and a revision of it (cf. Figure 6.7).

Figure 6-7 Summary of Communicative Characteristics of the Three Example Fast Commit Cases

Communicative Characteristics	Case 84	Case 78	Case 76
<u>Communications (Total)</u>	7	5	8
Communicative	6	3	6
Discursive	0	1	1
Instrumental	1	1	1
<u>Influence Tactics (Total)</u>	1	2	1
Rational Persuasion	1	2	1
Consultation	0	0	0
<u>Validity Claims Challenged (Total)</u>	0	1	1
Comprehensibility	0	0	0
Effectiveness	0	1	0
Efficiency	0	0	1
Relevance	0	0	0

Finally, Case 76 is a New Task innovation type of high complexity proposed by Moshe Weitzman. There are four other participants in this SRD process, Stefan Nagtegaal, Chx, Berkes Kessels and Dries. Moshe proposes his innovation discussing in detail its purpose and functionality:

January 26, 2005 - 14:18 : moshe weitzman

Attachment: <http://drupal.org/files/issues/drdest.patch> (11.05 KB)

Here is a patch I've been wanting to finish for a while. This patch assures that you end up on the proper page after you edit/delete a node, comment, user, or url alias. This is true no matter if you go through the usual interface or the admin interface. Further, if click the 'edit' link from 3rd page of a custom sorted view (e.g. admin/comment&from=100&sort=asc&order=Author) you still are returned to the right page.

The technique used here is generally available for module developers. I've minimally enhanced drupal_goto() so that it will redirect to the url specified in a 'destination' querystring parameter if such parameter exists. If it does not exist, we redirect just as today. No changes are required to existing drupal_goto() calls. A new helper function, drupal_get_destination() was added; it helps construct the 'destination' string which is appended to add/edit links.

The only downside I can see to this patch is that a few URLs are less pretty than before. These urls are only shown to admins. This could only be avoided by having each admin page implement its own way of passing a destination, or stashing the destination in the \$_SESSION. We recently tried storing referer in \$_SESSION, and it was eventually removed because of poor coordination when a user has multiple browser windows open.

In addition to the above,

- I cleaned up some 'destination' handling in user login code*
- I assured that after adding a new taxo term, we arrive back on the 'Add' page. That restores prior behavior*

Stefan responds in support of the proposed innovation, voting it ‘++ this patch in HEAD’ and gives the following explanation;

January 26, 2005 - 16:44 : stefan nagtegaal

This is another great improvement when we look at usability! Moshe, you did a terrific job on this..

After this patch is applied every submitted page drupal_goto()'s the page you expect it to go..

This is really one of the best patches i'd seen and tested lately, so

++ for this patch in HEAD..

Dries then responds and requests additional input from the Drupal community. He states; “The code looks good, the functionality is handy but I'd like to hear other people's thoughts on this.”, Chx then enters the deliberations in support of the innovation proposal, stating; “great one. +1”. At this point Berkes Kessels, after scrutinizing the software code he makes some remarks, supports the innovation proposal but also offering his opinion on how to improve the patch. He states;

January 27, 2005 - 09:05 : B?r Kessels

It never really bothered /me/ that I was redirected to odd places, since I have a drupal-sitemap printed in my head ;). However, asking some clients, learned me that this patch would be greatly appreciated.

+1 from me.

One question though (not criticism!) why did you choose to do the testing inside druopal_goto as

<?php

*if (\$destination = \$_REQUEST['destination'] ? \$_REQUEST['destination']
: \$_REQUEST['edit']['destination']) {*

?>

Seems odd to me to have a one-line-if inside another if.

Dries picks up on the challenge and offers a revision to the software code of the innovation proposal. He states:

January 27, 2005 - 09:18 : Dries

Berkes: that line is a odd, indeed.

Actually, I'm not convinced that embedding this logic in drupal_goto() is appropriate. Personally, I'd rather have us write:

```
<?php  
  
drupal_goto($_REQUEST['destination']);  
  
?>
```

I'd like to believe it is more transparant.

Moshe then revises the software code of his proposal and Dries legitimizes it stating; “Committed to HEAD. Thanks!” and assigned Moshe to implement the software update. The entire deliberation comprised seven communications of six communicative actions and one discursive challenge to the efficiency of the software code.

The SRD legitimization processes of these three examples of Fast Commits are indicative of this category. The majority of proponents were able to achieve success in legitimization of their innovation proposals by presenting them to the community and refining them based on the community’s feedback. In other words, as opposed to some of the Slow Commits which involved the collaborative design efforts of the Drupal community, the innovation proposals in this category had a single proponent who designed and refined the innovation after obtaining feedback.

6.2.4 An Example of a Slow Commit SRD

I will now briefly discuss an example of a Slow Commit to illustrate the communicative dynamics of this category of innovation proposals. The empirical observations used in this illustration were derived from the coding and content analysis of SRD communications on Case 12. The type of innovation proposal concerned is a Bug Fix of high complexity, and there are a total of 9 participants in the SRD and a total of 49 communication exchanges. Of these 49 communications 39 are communicative actions, 6 are discursive actions, and 4 are instrumental actions. During the SRD cycles rational persuasion was the most dominant influence tactic enacted (18 instances) and Consultation was also used (6 instances). See Figure 6.8 below for a summary of the primary communicative characteristics of slow commit Case 12. In the following I present a portion of the 49 communications in context to illustrate the dynamics of the SRD.

Figure 6-8 Communicative Characteristics of Slow Commit Case 12

ISS Dimensions	Slow Commit (Case 12)
<u>Action Types (Total)</u>	49
Communicative	39
Discursive	6
Instrumental	4
<u>Influence Tactics (Total)</u>	24
Rational Persuasion	18
Consultation	6
<u>Validity Claims Challenged (Total)</u>	5
Comprehensibility	0
Effectiveness	2
Efficiency	3
Relevance	0

The SDR for Case 12 started with Pennywit employing a *communicative action* to make a bug report on October 7, 2004, 15:23;

October 7, 2004 - 15:23 : pennywit

I have two problems with my upgrade at <http://www.pennywit.com>. First is that a number of comments have the body turn to the number 3. It looks like this happens when one of my users tries to edit his comments. The second is that in any nodes submitted before I upgraded, I don't see a count of the comments already submitted. I'm echoing this to the support forum ...

Pennywit later uses a consultation tactic in the following comment requesting information. This comment is communicative in type

October 7, 2004 - 16:01 : pennywit

It says this has been fixed ... what do I need to do on my side?

--/PW/--

The next day (October, 8, 2004-09:11) Kessels responds with his rationale that that problem has already been fixed elsewhere, and provides a link for the solution. This is a *communicative action* pointing Pennywit to a potential solution;

October 8, 2004 - 09:11 : B?r Kessels

<http://drupal.org/node/11316> was the bug. Update comment.module will fix this problem.

However, a few days later (October, 17, 2004 -13:04) KPS enters the SRD pointing out that problem persists (*discursive action*), challenging the effectiveness of Kessel's solution:

October 17, 2004 - 13:04 : kps

*This problem remains (in 4.5.0-rc 1/2 hour before this post):
... any nodes submitted before I upgraded, I don't see a count of the comments already submitted.*

The {node_comment_statistics} table is evidently not initialized correctly.

Justification for "critical": Since most people who install 4.5.0 will be users of older releases, upgrading really ought to work before the release.

Less than an hour later KPS exerts a rational persuasion influence tactic by attaching a patch of his own, directed at solving the problem at hand:

October 17, 2004 - 13:47 : kps

*Attachment: <http://drupal.org/files/issues/updc.php> (1.26 KB)
Attached, raw and as-is, the script I used to initialize the {node_comment_statistics} table.*

A few more days go by before anyone else comments on the reported problem. Then Junyor enters the SRD (October, 20, 2004 – 08:39), supporting KPS's observations. Junyor further explains the problem and provides some recommendations to solve the problem at hand:

October 25, 2004 - 08:39 : Junyor

kps is right, the update won't correctly populate the table. Here's (part of) update_105:

*"INSERT INTO {node_comment_statistics} (nid, cid,
last_comment_timestamp, last_comment_name, last_comment_uid, comment_count)
SELECT n.nid, 0, n.created, NULL, n.uid, 0 FROM {node}n"*

*So, it's setting the comment_timestamp = node creation time, forgetting the
comment_creator, using the userid that created the node as the*

last_comment_uid, and setting the comment count to 0. update_105 needs to be redone and I'd recommend a new update be created for 4.5.1 that will correctly initialize the node_comment_statistics table.

Several days later (November 12, 2004 - 10:02) Junyor suggests a new Bug Fix to the Drupal community:

November 12, 2004 - 10:02 : Junyor

OK, here's two patches to solve the problem.

Issues fixed:

- Fixes node_comment_statistics table prefixing for PostgreSQL*
- Drops CID column from node_comment_statistics and supporting code in comment.module*
- Initializes comments table with usernames (needed by node_comment_statistics table)*
- Correctly initializing node_comments_statistics table*
- Removed duplicate query for last_comment_name in node_comment_statistics query in comment.module*

Needs checking:

- I couldn't test this with PostgreSQL*
- I'm no database expert, so the updates.inc changes need to be gone over with a fine-toothed comb*
- Do we need any special handling for comments with no 'name', i.e. truly anonymous comments*

These patches are for the DRUPAL-4-5-0 branch.

Junyor provides two more relevant patches which he discovered in the Drupal repository:

"November 12, 2004 - 13:01 : Junyor

Attachment: http://drupal.org/files/issues/node_comment_stats.patch (7.89 KB)

November 12, 2004 - 13:01 : Junyor

Attachment: http://drupal.org/files/issues/node_comment_stats2.patch (3.44 KB)

Actually attaching patches. :)"

Dries then enters the SRD (on November 14, 2004 - 21:10) using *discursive action* to challenge the efficiency of Junyor's solution. He expresses his reservations and suggests further testing before the Bug Fix is legitimized and implemented into the Drupal core software. He states:

November 14, 2004 - 21:10 : Dries

I'm not 100% sure but isn't the name field of the comments table supposed to be NULL, unless the comment is an anonymous comment? Is it really required to initialize the name field? What happens if you don't? I just checked drupal.org's database and only post Drupal 4.5.0 comments have the registered user's name in the comment table.

I'm a little nervous about committing database changes to stable branches (DRUPAL-4-5) so please make sure this patch is well-tested.

The patches don't apply against HEAD but I can port them once we/you ironed out the last glitches.

Otherwise these patches look fine. Good job.

To this Junyor responds (November 14, 2004 - 23:41):

November 14, 2004 - 23:41 : Junyor

I don't know if it's necessary, but I'm just making sure it's consistent. Maybe the author of the node_comment_statistics patch could comment? I'm also concerned about having the name information for registered users outside of the users table.

I've tested this on a test site and I'll try testing on my production site once we have this worked out.

To which Dries responds with another discursive action challenging the efficiency of the same patch:

November 15, 2004 - 10:45 : Dries

I don't know whether the original author (ccourtne) is still around but it should be easy enough to test whether it is necessary or not. From what I've seen, it isn't necessary. In fact, your current patch might break things: like, what happens when someone changes his or her username? AFAIK, the old username would be shown in the comments.

Junyor then responds with *communicative action* (November 15, 2004 - 11:03):

November 15, 2004 - 11:03 : Junyor

OK, I'll edit that bit and resubmit. You'd like this for HEAD only?

Junyor continues to develop the solution to the Bug Report and later submitted four software patches for consideration:

"November 15, 2004 - 18:24 : Junyor

*Attachment: http://drupal.org/files/issues/node_comment_stats-2.patch (7.24 KB)
New patch for database files.*

November 15, 2004 - 18:25 : Junyor

*Attachment: http://drupal.org/files/issues/node_comment_stats2-2.patch (3.46 KB)
New patch for comment.module.*

November 22, 2004 - 11:31 : Junyor

*Attachment: http://drupal.org/files/issues/node-comment-statistics_head.patch (7.26 KB)
Patches for head.*

November 22, 2004 - 11:31 : Junyor

Attachment: http://drupal.org/files/issues/node-comment-statistics_head2.patch (3.45 KB)"

After more experimenting and testing Dries invites further participation from other members of the Drupal community :

November 23, 2004 - 23:24 : Dries

I'd like to move forward with this patch and include it in Drupal 4.5.1. I can't reproduce this problem (it seems) so it would be much appreciated if those who can, can test it.

Some days pass and other members of the community respond, except for Junyor who again tries to use rational persuasion to convince Dries and the community of the effectiveness of his software innovation. He says:

November 29, 2004 - 00:42 : Junyor

It applied and works well on my 4.5.0 site. A lot of nodes were listed as not having comments before, but they're working now. Dries alerted me to a drupal-support message regarding the patch and I'll look into that tomorrow.

Then finally CRW enters the SRD using discursive action and challenges the effectiveness of Junyor's proposal. CRW states:

December 8, 2004 - 06:12 : crw

After applying the patch, approving new comments still does not update the node_comment_statistics table, therefore the comment_count field is off and the number of new comments does not show up on the main page of my site. I'm currently troubleshooting this and will report my findings here.

December 8, 2004 - 06:13 : crw

I should point out that I applied the following patch: node-comment-statistics_head2.patch And the problem described above still exists.

After some more testing and experimentation CRW appears to have found a solution to the problem and makes a new proposal. Since CRW is posting a patch of his own, based on his reasoning, thus he exerted a rational persuasion influence tactic:

December 8, 2004 - 06:25 : crw

Ok, headway.

Looks like submitting a comment triggers an updating of node_comment_statistics, but editing a comment to publish/unpublish does not. Both should trigger the update, so I just need to find out why it isn't working for the 'update' case.

December 8, 2004 - 07:22 : crw

Ok, I found the problem and came up with a solution. I've been manually approving comments from anonymous users via the admin interface. comment_admin_edit() calls comment_save(), but neither calls _comment_update_node_statistics. I inserted the following two lines around line 952 of comment.module after the call to comment_save():
\$nid = db_result(db_query('SELECT nid FROM {comments} WHERE cid = %d', \$edit['cid'])); _comment_update_node_statistics(\$nid);
Ideally, there'd be tests for all these. comment_save should produce a return value, and comment_admin_edit() shouldn't go forward unless comment_save returns true.
Please excuse my lack of diff/patch-fu. :)

Junyor responds some hours later and challenges CRW's proposal. He states:

(December 8, 2004 - 09:58): Junyor

"Bah, that function should call comment_save(). All comment changes should go through comment_save(). It would make life so much easier.
I'll try to come up with an updated patch soon, but probably not before this weekend."

Dries then responds in support of Junyor and again ask for input from other Drupal community members. He states:

(December 8, 2004 - 10:28): Dries

"I agree that all comment editing should go through comment_save(), yet that will require a bit of refactoring. Also, there are two 'edit comment' forms (one for users, one for administrators) that should be merged, much like we merged 'node edit' forms.
Anyone?"

A few more days pass then Junyor follows up with Dries on the status of the Bug Fix.

(December 11, 2004 - 17:19): Junyor

"Dries: I see that you made a change to the updating of node_comment_statistics in CVS HEAD. Do I still need to add an updated patch? Are there plans for a 4.5.2 that could use this patch?"

An hour later someone new, Ax, enters the SRD. Ax reports on his own problems with the Bug and his experimentation with Junyor's Bug Fix. He supports Junyor's proposal but has some comments regarding the effectiveness of the patch in solving the problems and points out that there are some other software modules which are affected. Ax says

(December 11, 2004 - 18:14): Ax

"i tried upgrading a 4.4 site to 4.5 yesterday and fell into the same trap as pennywit, kps, junyor, and others: the update for the node_comment_statistics table only updates forum comments and inserts num_comments 0 for all other node types. quite cheaty, this. junyors 4.5 patches (node_comment_stats-2.patch, node_comment_stats2-2.patch) seem to solve the problem for me. At least, i see proper "replies" counts in tracker and node views now. I didn't try approving / publishing / unpublishing comments, though, nor did i check other issues mentioned in the thread (user names postgresql, ...). i applied the 2 patches to a 4.4 database / 4.5 code both before running update.php and afterwards. in the first case, there is a small glitch in that update.php throws an error:

user error: Unknown table 'node_comment_statistics' DROP TABLE {node_comment_statistics} because the patch removes "CREATE TABLE {node_comment_statistics}" from update_105. this could be caught by changing to "DROP TABLE IF EXISTS {node_comment_statistics}". this is definitely a critical bug, and these patches should be applied to 4.5 as soon as possible, regardless of the "all comment editing should go through comment_save()" issue."

Another new entrant to the SRD, Jeremy also raises a challenge to the effectiveness of Junyor's proposal. Jeremy states

(December 11, 2004 - 20:53): Jeremy

"I've just run into this same bug, trying to upgrade from 4.4 to 4.5. It's a show stopper for me. :(Ugh, so close."

The SRD continued with more rounds of communications with three other participants joining the deliberations until they were able to resolve the issues. Dries then legitimized the revised and tested innovation proposal and assigned Junyor to implement it. In his final communication on this issue Dries says:

(January 5, 2005 - 21:37): Dries

"I committed the patch to DRUPAL-4-5. I'm putting the HEAD version on hold until the revisions patch landed. Please upgrade your Drupal 4.5 sites to DRUPAL-4-5 in preparation of Drupal 4.5.2. Thanks."

6.3 Concluding Summary

In this chapter I set out to illustrate the basic characteristics of the Structured Rational Deliberations which I have hypothesized serve as the legitimation process for new innovation proposal in the Drupal open source community. In this section I want to summarize some of the key empirical observations from my data analysis. (1) The participants of the SRDs had a preference for communicative action in all cases. By that I mean that communicative action dominated the deliberations of committed as well as uncommitted innovation proposals. Discursive actions were used as a means of challenging the validity claims of an innovation proposal and vetting the quality of it. Very few instances of instrumental action have been observed in the data analyzed. (2) The participants of the SRDs had a preference for rational persuasion. This influence tactic was used by the participants to influence each other in the following two distinct ways: (a) defending a validity claim of the innovation (eg. relevance); and (b) in obtaining support from Dries or other key actors in Drupal. (3) The SRDs were always open and anyone could participate. In many cases Dries (as well as other members) put out open requests for participation in the deliberations. (4) A proposal succeeded (committed) or failed (uncommitted) based on defense that the proposer could mount for it. Mounting a defense was not a blind process, it often required the proposer redesigning and improving the proposal based on feedback from the community.

7. Theoretical Discussion

The empirical analysis in the prior chapter illustrates how the innovators of Drupal used open discourse to cultivate legitimacy for their innovations in order to obtain approval from Dries for inclusion in the software. In this chapter I will offer a theoretical explanation of my empirical observations on the dynamics of Drupal's innovation legitimation processes from the perspective of the ISS theoretical framework I outlined earlier in Chapter 3. The primary focus of my discussion is to develop a theoretical explanation of Drupal's innovation legitimation process from the perspective of the concept of Situated Rational Deliberations (SRD) in the Ideal Speech Situation (ISS) theoretical framework. However, before moving to that discussion I must first provide a theoretical discussion of the social organization of Drupal open source community, which has a different social structure to traditional software organizations. Therefore two important issues require elaboration here: (1) the social and organizational context of Drupal and (2) the dynamics of the innovation legitimation processes.

7.1 The Social and Organizational Context of Drupal

The Drupal Open Source Development (OSD) community is a non-profit altruistic organization that is composed of members who donate their time and expertise to develop, innovate upon and maintain software products for the public goods economy (Bergquist & Ljungberg, 2001; Zeitlyn, 2003). Traditional profit-oriented software development organizations are different in that they have allocated resources (people, money, time and expertise) and a defined hierarchy of authority, roles and organizational policies and procedures oriented to their profit motive. Further, traditional software organizations have reward and incentive schemes for motivating and rewarding appropriate behavior of its members (Bitzer, Schrettl, & Schroder, 2007; Nelson & Winter, 1982). However, the dominant philosophy of OSD is the free creation of software for the public good, and anyone with the appropriate expertise and motivation is welcome to contribute (Bonaccorsi & Rossi, 2003; Kogut & Metiu, 2001). In this regard, the Drupal OSD community is no exception; anyone can submit new features, updates or patches to Drupal's core software (Drupal, 2013). There are no limitations on participation (proposing or criticizing innovation proposals) on any of Drupal's deliberation media, which are mainly the Issues Queue (online forums), Drupal IRC Channel (#drupal-contribute) and the developer mailing list

(Drupal, 2013). Further, the process for legitimizing an innovation proposal is open deliberative discourse in which no limitation exists on who can comment during the deliberations. The empirical observations from the 52 cases showed that for an innovation to be committed into Drupal software the innovator must defend it against the scrutiny of the community. When a Drupal member proposes an innovation for the software environment it is subjected to open critique and approval is given only when the innovation receives support from a significant number of the community members (ie. a ‘+’ rating). From this perspective Drupal’s innovation legitimization processes can be viewed as a set of Situated Rational Deliberations and a voting process. During the deliberation process Drupal members may criticize an innovation proposal by: (1) challenging its validity; (2) suggesting changes to the innovation or offering alternatives; or (3) outright rejection of the idea. This process will be discussed in more detail later in this chapter.

Participation in the rational deliberations is highly valued and critical to the viability of Drupal. The social structures of the Drupal community orients its members and endow them with rights and responsibilities of participation in these SDRs that are the primary mean of critically interrogating innovation proposals and achieving the goal of high quality software, which is at the heart of Drupal’s philosophy. All “contributions are peer reviewed and then decided on by Dries or another of the core committers” (Drupal , 2013). Further, status and recognition in Drupal is achieved based of the contributions that members make to designing and programing innovations, and engaging in deliberative discourse to improve the value of Drupal’s software core capabilities.

7.2 Leadership, Roles and Rewards

Within the OSD community of Drupal there are four definable roles and responsibilities: (1) **core committer (Obligatory Passage Point)**, (2) **maintainer**, (3) **branch maintainer**, and (4) **core contributor**. All roles defined in the Drupal project, except for OPP, are dynamic and changing dependent upon continued participation in the Drupal community. Role fluidity was also observed in other empirical studies of OSD projects conducted by (Tirole & Lerner, 2002). However, this does not mean that OSD organizations are unstructured, anarchistic cultures that lack leadership; on the contrary they usually have strong community values and strong leaders

(Bonaccorsi & Rossi, 2003). Often, the OSD community is started by a few individuals who often rotate the leadership responsibility for the group; and in some cases the leadership role is decided by community voting (O'Mahony S. &, 2007; O'Mahoney, 2008). In the case of Drupal, Dries is the singular visionary who started the project of building the software and as the community grew he assumed the role of OPP, encouraging and monitoring the deliberations and giving final approval to innovation proposals after a significant number of members have voted. Dries is also responsible for appointing community members to the various roles described above (Drupal, 2013); on this account, I classified him in the role of OPP.

While Drupal's environment might seem unstructured and unconstrained due to its social organization, it is far from that. As found in Linux, Drupal has an undisputed leader, who voices 'recommendations', and even though the leader does not have the authority to obligate any member to tasks these 'recommendations' are apparently followed by the majority of the community. This is mainly because in open source projects there is certain trust between the leader and the community that stems from the notion that the community will follow the 'recommendations' of the leader only if the leader respects the community's input into the work conducted (Tirole & Lerner, 2002). In other words, the recommendations will be translated into code only if the community members feel that it is in the benefit of the software, and not an ego-centric order from the leader. This also means that in order for the leader to be respected, he/she must be able to entertain criticism and improvements to their recommendations (Tirole & Lerner, 2002).

The three (2-4) basic roles are conferred by Dries, and are temporary, based on a member's continued contributions to the Drupal software environment. This too is a common practice in OSS communities; role specification results from the initiative of the volunteer (O'Mahoney, 2008; Krogh, 2003). At the time of this research there was 1 *core committer*, which was Dries, 10 branch *maintainers*, whom did not differ in responsibilities from 'maintainers', and an unknown number of core contributors. A Core Committer has the responsibility of testing and reviewing patches, writing tests, patching issues and documentation (Drupal, 2011). Dries is the only permanent Core Committer and he is the only member that has access to all the software code in the core depositary and the final authority in approving any changes to the software

(Drupal, 2013). The *Maintainer* has “informal responsibility” for a certain portion of the software core (Drupal, 2013). The next role of core contributor is anyone who contributes to the Drupal core software. As stated in Drupal.org, “Core contributors who have made substantive contributions (particularly to a core component not individually maintained) may apply for Maintainer status by writing to Dries. Dries may also individually invite them” (Drupal, 2013). In addition to the *Maintainer* role, Dries also appoints the *Branch Maintainer* role to members that have contributed substantially to certain modules of the Drupal core software. A *Branch Maintainer* has **update** access to the core repository and is allowed to commit a patch, but only to specific modules of the software (Drupal, 2006). The appointment of *Maintainer* and *Branch Maintainer* is based on a member’s past contributions to the core software. While, any community member can ‘apply’ for a formal role, such ‘applications’ are only successful if the individual has a track record of contributions to the community. This observation is corroborated in the literature by Ye (2003) and O'Mahony (2007) whom show that positional authority is conferred in recognition of contributions to the community. In another report Bonaccorsi and Rossi (2003) state that in open source software roles and authorities tend to grow naturally with the software and are assigned on the basis of contribution to the code, as opposed to being assigned from the beginning of the project.

Contributors receive no direct materialistic or monetary gain from contributing to the Drupal OSS community. There are two types of rewards that are conferred on active long term contributors to the Drupal community: (1) appointment to a specific role; (2) public recognition of contributions in the Drupal member directory. The appointment to a role signals recognition of continued commitment to the community and confers upon the appointee social capital and status within the community (Bergquist & Ljungberg, 2001). These appointments also have a value outside of the community as they signal to external organizations the competence and expertise of the person. Contributions to successful Drupal initiatives has the potential to boost an individual reputation, which in itself is a reward as it has the potential to give credit beyond the Drupal community (Ljungberg, 2000; Zeitlyn, 2003). In the context of the Drupal project, contributors gift their time and expertise, which result in a piece of code or a certain approach to address a problem, and receive in return the credit and recognition of being the original contributor of that piece of code or approach. The more popular the innovation becomes the

more recognition the original contributor of that innovation will receive. Also, in each member's profile in Drupal.org a counter measures the number of successful initiatives credited to each member; the higher that counter, the more power that member's opinion holds in future discussions. Giving public recognition to a member for an innovation is an essential practice in OSD (Tirole & Lerner, 2002). In the Drupal community each member's profile lists all innovations that member was responsible for, along with all projects and community events that members attend.

7.3 The Dynamics of Structured Deliberations

While there are specified formal roles in the Drupal community the legitimization of innovation proposals unfold as a set of open structured deliberations within a social structure that allows for any of its members free and open participation. The empirical observations suggest that the process for structured deliberations on innovation proposals at Drupal closely approximates the Ideal Speech Situation (ISS) (Habermas, 1970), with its basic premise being the following:

- all participants have equal opportunity to engage in discourse
- all participants have equal opportunity to freely voice their opinion, and to challenge others', about any discourse in the context of the deliberations
- the discourse is free of constraints such as domination, manipulation and control
- all participants' input must be equal in power

Empirical observations from the 52 cases of structured deliberations on innovation proposals show that none of these basic criteria were violated. The structured deliberations took place in Drupal's open forums, its public communication system, to which all of its members subscribe. At no time were any of the communications private (I have no way of knowing if they were back channel communications between any of the members). At no time was any one admonished for being overly critical of an innovation proposal; on the contrary, at times Dries often requested more critical discussion on a proposal if there seemed to be a lack of interest in the discussion. At no time was discussion shutdown or prevented from unfolding. Contributors had the option to defend, modify or withdraw their innovation proposals in the light of the unfolding deliberations. Other participants in the deliberations had the option to criticize, suggest modifications to or vote

for all innovation proposals under consideration. While Dries, the OPP, monitored the deliberations and observed the voting he for the most part kept silent, allowing the Drupal members to speak freely. These systematic empirical observations on the patterns of the structured deliberations suggest the following theoretical proposition:

P1: When the social organization is heterarchical and the participants are oriented to design excellence, their deliberations on innovations are likely to be open and unfettered.

This theoretical proposition is corroborated by the literature on design and policy studies. For example, Wylant (2008) found that innovation flourished in organizational situations where open discussions among designers are incentivized. Simons (1999) also found that top management teams were more effective in strategic decision making when they encouraged diversity in ideas and debate. Van Der amd Schoemaker (1992) illustrated the importance of open communication to solving deep and seemingly intractable problems. Tjosvold, Tang and West (2004) argue that open discussion improves a team's problem solving capability and leads to better performance outcomes. On the other hand, Marx (1991) illustrated how restricted discussion led to poor strategic decisions in firms. Moreover, an early software development study by Ngwenyama (1991) illustrated how modelling software design processes along the lines of ISS improved collaborative action learning of software designers and led to better outcomes.

7.4 Communication Dynamics of Structured Deliberations

As observed in the empirical data, Drupal's innovation legitimation process unfolds as open SRDs in which Drupal community members critically interrogate each innovation proposal for relevance, efficiency and effectiveness. The proponent of the innovation must present a thoughtful argument for the innovation and defend it to his/her peers. The deliberations are conducted in Drupal's online forums and the innovation is only legitimized by Dries after extensive critique and ascent of the community members. During the deliberations individuals use rational persuasion and consultation techniques to influence each other on the suitability of the proposed innovation for inclusion into the core software. My analysis of the conservations reveals that both the innovators and critics try to indirectly influence Dries, the OPP, for all changes to the core software. However, all Drupal community members understand that Dries is

not the only one to persuade, so a large network of influence can be observed in the conversations.

When the structured deliberations on innovation proposals are examined from the perspective of the ISS framework a pattern of communication is observed that is of importance to our developing theoretical understanding of innovation legitimation processes in heterarchical organizations. First I observe that the structured deliberations are dominated by communicative actions; that is the orientation of the participants in these deliberations is in reaching understanding. Second, the focus of critique of innovation proposals is on interrogating and testing the implicit validity claims of efficiency, effectiveness and relevance of innovation proposals. Third the preferred strategy for influencing each other was rational persuasion and consultation. These patterns of communication are very different from traditional software development organizations. The empirical observations from the analysis of the conversations suggest the following proposition:

P2: When the social organization is heterarchical and the orientation of participants is on building shared understanding for joint action, the participants will more likely use communicative action and rational persuasion to influence each other.

The literature on communication and organizational influence support this finding. For example, Innes & Booher (1999) discussed the central role that communicative action plays in consensus formation in complex adaptive systems. Polanyi (2002) found that communicative action is an efficient approach to building common ground and achieving consensus for joint action in large group collaborations. Ngwenyama and Nielson (2013) found that when individuals in software projects need the support of others over whom they have no authority they use soft influence tactics of rational persuasion and consultation to enroll their support. Yukl, Falbe and Youn (1993) conducted an empirical study on the effectiveness of different patterns of influence by managers and suggest that rational persuasion and consultation are more effective forms of influence when dealing with peers. Sussman and Vecchio (1997) found that individuals choose different influence tactics dependent upon their position in the organization relevant to the target of influence. Schriesheim and Hinkin (1990) also argue that rational persuasion is a more effective influence strategy when dealing with peers. Other studies have found that organizational members respond more favorably to the use of soft tactics because such tactics

emphasize trustworthiness and gives credibility to the target of influence (Barry & Shapiro, 1992). Furthermore, an organizational member's behaviour depends on what they perceive to be appropriate in a certain social setting (Cartwright & Zander, 1968). For example, individuals working in settings of participative management tend to exercise rational communication and rational persuasion, while individuals working for authoritative and hierarchical management settings, often choose non-rational tactics such as blocking and upward appeal (Ansari & Kapoor, 1987). Also, in a study measuring the effectiveness of different influence tactics, Falbe and Yukl (1992) found that the rational persuasion influence tactic was much more effective when used in combination with consultation.

7.5 Characteristics of Legitimation Processes

Empirical observations from the data analysis revealed two distinct categories of innovation legitimation processes which I earlier referred to as Fast and Slow Commits. From a design perspective, one distinguishing feature of Fast and Slow Commits is the complexity of the innovation proposal and its potential implications (impact on other software code and future maintenance) for the Drupal software environment. Therefore, complex innovation proposals require significantly more deliberations and receive a higher frequency of challenges to efficiency claims, resulting in slower (or longer) legitimation processes. Of the 24 legitimized innovations in the empirical analysis, 16 are characterized as the Fast Commits, and 8 as Slow Commits. Empirical findings from the content analysis of the 24 successful legitimation cases illustrate the low complexity and straight-forward nature of those 16 innovation proposals characterized as fast commits, while a much higher level of complexity is apparent for the 8 deliberations characterized as slow commits (Table 6.4). On the average the deliberation cycles for slow commits were slightly more than 3 times as long as fast commits (5.3 and 17 comments per case for fast and slow commits, respectively). Further, there are 11 instances of validity claims challenged in fast commits, as opposed to 18 such instances in slow commits (mostly challenging efficiency claims). Also, fast commits consisted of 37% bug reports, 25% new features, and 37% new tasks, while slow commits consisted of 63% bug reports, 25% new features, and 13% new tasks. (cf. Appendix E)

A fundamental difference between the two categories was that redeeming the validity claims took more effort in slow commits than in fast commits as the innovator was often required to convince opponents of the relevance, effectiveness, efficiency and implications of the proposed innovation and in many cases are required to make modifications to it. When bug fixes, new features, and new tasks are compared in terms of communicative action types, it becomes apparent that out of those three types of innovations, bug fixes required the longest deliberations. Empirical analysis of the deliberations on new features showed that, unlike bug fixes, they do not entail modifying or eliminating existing code; rather they add features to the software with little downstream impact. This allowed the deliberations to take a simpler form, and receive less criticism from less people. It seems in the data that when attempting to address bug reports, innovators needed to change some aspect of how a certain piece of the software operated, thus dealing with the risk of change. In addition, even though Drupal operates on a modular design, changing one aspect of the core software can potentially result in inconsistency or interchangeability issues with the rest of the software. Facing one of these issues as a result of addressing a bug report can ultimately deem a fix more expensive (in terms of resources). Thus, in an attempt to adopt the most effective approach, Dries is observed to encourage a longer deliberation in cases that address bug fixes.

P3: The type, complexity and implications of the proposed software innovation will influence length and character of legitimization process.

P3a: When the proposed innovation is complex and has the potential for complex impacts on the core modules the legitimization cycle will be long, as the innovator must defend the relevance, effectiveness, efficiency, reputation and cost implications of the proposed innovation to the OSS community.

P3b: When the proposed innovation is in response to critical bug fixes the legitimization cycle will be long, as the innovator must defend the effectiveness, efficiency, reputation and cost implications of the proposed innovation to the OSS community.

The literature on software design and design theory support this finding. For example Waterman (2013) have found that complexity of the software, that is, when functionality is high software design is more costly in time and effort. Martin (2003) also argues that the more variety in the functions of the software the higher cost to develop in terms of time and resources. This finding about the cost of complexity has also been observed in other design domains such as product design and manufacturing (Banker, 1990). Another dimension of importance here is organizational structure and values in open source development (Crowston, K, Wei, Eseryel, & Howison, 2007). Communication processes in virtual organizations are more time consuming for several reasons (Kasper-fuehrera, 2001; Khalil, 2002). First, communication is asynchronous as users are not face-to-face, this can lead to time delays in response and breakdowns in communication which must be repaired (Bjorn & Ngwenyama, 2009). Members of virtual organizations need to develop specific steering processes to ensure smooth and effective communication (Bjorn P. N., 2010). Further, significant effort must be spent on overcoming cultural differences and building trust when the team members are multi-cultural (Gallivan M. J., 2001; Soderberg, 2013). Also, when the organization is oriented to maintaining its reputation much more effort is put into making the best decisions concerning their products (Arvai, 2001; Dijksterhuis, 2006). Reputation is highly valued in open source communities; it is what attracts volunteers to donate their time and expertise to the OSD project (Stewart & Gosain, 2006; O'Mahony S. , 2007). The orientation and identity of OSD volunteers is towards excellence in software development (Blohm, Bretschneider, Leimeister, & Krcmar, 2011)

7.6 Use of influence Tactics in Structured Deliberations

Another important empirical observation here concerns the characteristics of the use of influence processes during structured deliberations of innovation proposal. The organizational and social norms of Drupal dictate that all innovation proposals be subjected to a collaborative code review process and each proposal must obtain a majority of positive votes to be considered for inclusion into the software platform by Dries. Consequently, in order to succeed the innovator must build a persuasive case for the innovation and recruit support from a critical mass of the Drupal community. But since the organization is heterarchical, and its members enjoy no formal authority over each other or Dries (the OPP), innovators who wish to get their innovations

legitimized and included in Drupal core software have no option but to exert influence via the process of structured rational deliberations. Based on the observations made in the empirical analysis, members of the Drupal community realize that the organizational norms under which they operate dictates that if they wish to legitimate an innovation, they must offer a persuasive argument for it, while the rest of the community, including Dries, scrutinize it. Whether or not the innovation can survive or adapt to the community's scrutiny will dictate the innovation's fate. Moreover, Dries established collaborative code review as an essential process in the legitimation of innovation proposals. This process usually starts when a member of the Drupal community 'consults' openly or proposes an innovation that aims to address a specific problem in the core software. The act of starting a conversation about a prospective innovation in the Drupal forum is in essence an act of consulting the rest of the community. The deliberations then unfold as a combination of the following types of communicative interactions:

- the innovator announcing and describing the proposed innovation
- members accepting the validity claims of the proposed innovation
- members challenging or rejecting the validity claims of the proposed innovation
- the innovator defending or shoring up the validity claims of the proposed innovation
- members presenting alternative approaches to ones already presented
- the innovator modifying the proposed innovation in response to community input
- Dries outlining outstanding legitimation processes needed
- Dries accepting the proposed innovation for inclusion in the core software

Empirical analysis of the 221 communicative interactions during the structured rational deliberations of the 24 successful proposals reveal that on average 4.8 community members participated in reviewing each innovation proposal. More specifically, that average is 3.7 contributors in fast commits and 5.9 contributors in slow commits. Within the 24 conversations, Dries commented only 44 times, 25 of those comments were "committed", and 9 comments were requests for further input from the Drupal community. While Dries is the OPP for all innovation proposals, he manages the Drupal project to encourage rational deliberations amongst the community members, however from the innovator's perspective Dries is a key target of influence. Given the heterarchical organizational structure, the successful innovator must skillfully use soft influence tactics to recruit a critical mass of support from the Drupal

community in order to influence Dries to legitimate and accept the innovation. Empirical analysis reveals that rational persuasion and consultation are the dominant influence tactics used in recruiting support in the Drupal community. The process almost always starts with the innovator identifying a problem in the software and presenting a solution in the form of a patch (out of the 24 cases, this happens 20 times, as opposed to 4 instances where conversations were started with a direct consultation). It is empirically backed however that as deliberations unfold, rational persuasion is used to achieve consensus or the appropriate number of ‘+’ votes on the proposal. Requesting and accepting feedback from the community members and reasoning about the details of proposals were used as strategies for recruiting further support. For example, in 6 cases where the innovators suggested an innovation and consulted the community for feedback, a total of 20 responses included 15 alternative proposals from different community members, deeming consultation a more effective tactics for when members have identified a problem but not a solution. Another outcome of using consultation as an influence tactic is the potential for a more collaborative design, in terms of more members contributing to the design of an innovation as a result of an initiator, but even in this situation the acceptance of a certain innovation will be ultimately be as a result of a rational persuasion tactic. A consultation tactic is used for when the goal is to attract attention to a problem, where as a rational persuasion tactic is used for when the goal is to convince the community of a solution.

The empirical analysis reveals that innovators have a preference for soft tactics (rational persuasion and consultation) when interacting with other members of the Drupal community. Thus, in Drupal, rational persuasion and consultation are the dominant influence tactics used within the structured rational deliberations.

P4: When attempting to get their innovations legitimized, innovators must recruit the support of other community members, whom have no obligation to abide by the innovators rationale, by exerting lateral influences using soft tactics

This finding is replicated by Kipnis, Schmidt and Wilkinson (1980) whom together found that to overcome resistance of co-workers change agents will use coalitions as an influence strategy. In this case we see members of the Drupal community influencing each other to get their

innovations approved by Dries. The process of forming a coalition includes recruiting support towards one's proposal within the organizational setting (Yukl & Falbe, 1990). Resorting to coalitions is helpful considering the lateral structure of the organization and peer relationships among the community members (Mechanic, 1962). Kipnis, Schmidt and Wilkinson (1980) also state that members of organizations who intend to introduce new ideas often use rationality tactics, which encompass detailed explanations of the idea being proposed, and the rationale behind the proposal. Innovators in Drupal solely use soft tactics when they implement their influence processes because they realize that they possess no direct authority over members of the community. In their study, Yukl and Tracey (1992) found that influence tactics need to confirm to organizational norms and the influencer's position in the organization. The innovators within the Drupal deliberations realize that the heterarchical structure of Drupal made the use of hard tactics inappropriate, mostly because the Drupal community members are not obligated to abide by any rationale, and certainly not by hard tactics. Thus in an attempt to maximize their chances of success, innovators always use soft tactics. Moreover, using hard tactics could result in resistance, resilience or retaliation; soft influence tactics are more effective in open communication where gaining trust is important (Drake & Moberg, 1986; Gattiker & Carter, 2010; Kotter, 2003). It is unlikely that the Drupal community will be supportive of a member that is perceived untrustworthy, and as O'Mahony (2007) and Crowston et al (2007) put it, trust is an essential value of OSD and community members do much to cultivate and sustain it.

8. Conclusion

A valid assessment of the research goal is important in evaluating the contribution of this thesis. As stated in the introduction, this thesis set out with the goal to arrive at an understanding of the legitimation processes of innovations in open source projects, but before propositions were made on legitimation processes, the open deliberations in Drupal was analyzed. Habermas's Ideal Speech Situation (ISS) was used as a framework to view the Drupal deliberations from, and the key aspects of the theory, Communicative Actions and Validity Claims were used together with, Influence Tactics concepts to code and analyze the Drupal deliberations. HyperResearch was used to code the deliberations, and to analyze the data. From the analysis, this thesis was able to identify two different categories of innovation legitimation process, Fast Commits and Slow Commits, each with different dynamics.

Before theoretical propositions were derived from the empirical analysis of the legitimation processes, some more general findings were outlined and empirically supported. This study found that when heterarchical organizations are oriented towards design excellence, the deliberations on the innovations are likely to be open. Also, when participants in heterarchical organizations aim to achieve shared understanding for joint action, the communicative action type and rational persuasion influence tactic are most likely to be used by the participants.

In regards to the dynamics of innovation legitimation processes, this study was able to find that legitimation processes of innovations in open source software were influenced by the type, complexity and implications of the innovations on the rest of the community. More specifically, innovation legitimation processes that constitute complex innovations that have potential implications on the rest of the software will be long, as the innovator must defend the innovation's validity claims. Also, when innovations discuss critical bug fixes the legitimation process will be long, as this type of innovation will require effort from the innovator to defend the innovation's validity claims.

Also, when attempting to get their innovations legitimized, innovators will attempt to influence other participants in an open deliberation by exerting lateral influences using soft tactics.

Since there are no studies in the literature that study legitimation processes of innovations within open source systems, the propositions of this thesis are original to the literature, and should be used as a foundation to build future research on.

8.1 Implications of this Thesis

This study also has important implications for stakeholders of the software production industry. Firstly, this thesis adds further understanding to the open source literature by offering an insight into the dynamics of innovating within open source communities. The management of innovation within open source communities can benefit from the findings of this study by gaining a further understanding on the dynamics of open source software communities. This understanding has the potential to offer insight that is transferable to other open source software production projects in an attempt to foster innovation.

Also, stakeholders in traditional software development projects can benefit from this study's insight by gauging their community's characteristics against Drupal's community, which is evidently capable of managing innovation.

8.2 Limitation

This study has some limitations that are worth noting. Firstly, had time not been a constraint in this study, and uncommitted cases were followed until either committed or rejected, further insight into the dynamics of legitimation processes in open source software could have been exposed. If a significant amount of uncommitted cases were eventually committed, then the sample size of the committed cases would have increased, this could have changed the dynamics of the legitimation processes exposed. On the other hand, if the uncommitted cases were eventually rejected, then the study could have explored dynamics of legitimation processes of failed innovation in open source software. Secondly, this study addresses a total of 52 cases from the Drupal open source project, a small percentage of the total cases available in the dataset. This was due to time constraints. If this study were to consider a considerably higher amount of cases, then the witnessed dynamics could have changed. Finally, this study addresses a set of cases specifically from one open source project, Drupal and although the findings in this study helps gain insight into the legitimation processes of innovations in other open source projects, applying

the research methodology of this study to another open source software project could result in different dynamics. The more software projects that are examined using this thesis's research methodology, the better insight provided. Analyzing more open source software projects will help gain further understanding on the innovation legitimation processes in open source software projects, and solidify the findings of this thesis.

8.3 Recommendation for Future Research

One potential future research project based on this study would be a more comprehensive empirical analysis of the uncommitted cases in this study. This could result in further insight into the legitimation processes of innovations in open source software. Such research could be conducted using the same research methodology and theoretical perspective of this study. Additionally, a more in-depth empirical study of the 52 cases from the perspective of power and influence theory has the potential to provide insights into the political and power dynamics of deliberations in the Drupal open source software community. However, such study would require further understanding of the social structure in Drupal, and also different theoretical frameworks from the one used in this study might be needed. In addition, future research could potentially address the legitimacy of the different roles within open source software, and how participants are selected for certain roles in open source software communities.

Bibliography

- Abrahamson, E. (1991). Managerial Fads and Fashions: The Diffusion and Rejection of Innovations. *Academy of Management Review*, 586-612.
- Ansari, M., & Kapoor, A. (1987). Organizational Context and Upward Influence Tactics. *Organizational Behaviour and Human Decision Processes*, 39-49.
- Arvai, J. G. (2001). Testing a structured decision approach: Value-focused thinking for deliberative risk communication. *Risk Analysis*, 1065-1076.
- Ashforth, B., & Gibbs, B. (1990). The Double-edge of Organizational Legitimation. *Organizational Science*, 177-194.
- Ashton, J., & Oakley, A. (1997). *The Gift Relationship: From human blood to social policy*. London: London School of Economics and Political Science.
- Banker, R. D. (1990). Costs of product and process complexity. *Measures of manufacturing excellence*, 269-290.
- Barry, B., & Shapiro, D. (1992). Influence Tactics in Combination: The Interactive Effects of Soft Versus Hard Tactics and Rational Exchange. *Journal of Applied Psychology*, 1429-1441.
- Bergquist, M., & Ljungberg, J. (2001). The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 305-320.
- Bernard, C. (1938). *The functions of the executive*. Cambridge, MA: Harvard University Press.
- Bies, R. J., Tripp, T. M., & Neale, M. A. (1993). Procedural Fairness and Profit Seeking: The perceived legitimacy of market exploration. *Journal of Behavioral Decision Making*, 243-256.
- Bitzer, J., Schrettl, W., & Schroder, P. (2007). Intrinsic motivation in open source software development. *Journal of Comparative Economics*, 160-169.
- Bjorn, P. N. (2010). Technology alignment: a new area in virtual team research. *IEEE transactions on professional communication*, 382-400.
- Bjorn, P., & Hertzum, M. (2006). Project-based collaborative learning: negotiating leadership and commitment in virtual teams. *5th conference on Human Computer Interaction in Southern Africa* (pp. 6-15). CHI: Cape Town.
- Bjorn, P., & Ngwenyama, O. (2009). Virtual team collaboration: building shared meaning, resolving breakdowns and creating translucence. *Information Systems Journal*, 227-253.
- Blake, N. (1995). Ideal Speech Conditions, Modern Discourses and Education. *Journal of Philosophy of Education*, 355-367.

- Blohm, I., Bretschneider, U., Leimeister, J., & Kremer, H. (2011). Does collaboration among participants lead to better ideas in IT-Based idea competitions? An empirical investigation. *International Journal of Networking and Virtual Organisations*, 106-122.
- Bonaccorsi, A., & Rossi, C. (2003). Why open source software can succeed. *Research Policy* , 1243-1258.
- Bonaccorsi, A., & Rossi, C. (2003). Why open source software can succeed. *Research Policy* , 1243-1258.
- Burger, T. (1976). *Max Weber's theory of concept formation: History, laws, and ideal types*. Durham, NC: Duke University Press.
- Burleson, B., & Kline, S. (1979). Habermas' Theory of Communication: A Critical Explication . *The Quarterly Journal of Speech*, 412-428.
- Callon, M. (1986). Some elements of a sociology of translation: Domestication of the scallops and the fisherman at St. Brieuc Bay. In J. Law, *Power, Action and Belief: A New Sociology of Knowledge?* (pp. 196-233). London: Routledge.
- Cartwright, D., & Zander, A. (1968). power and influence in groups. *Group dynamics*, 580.
- Carver, B. (2005). SHARE AND SHARE ALIKE: UNDERSTANDING AND ENFORCING OPEN SOURCE AND FREE SOFTWARE LICENSES. *Berkeley Technology Law Journal*.
- Clark, D. (1988). The design philosophy of the DARPA internet protocols. *Computer Communication Review* , 106-114.
- Cooren, F. (2000). Toward another ideal speech situation: A critique of Habermas' reinterpretation of speech act theory. *quarterly journal of speech*, 295-317.
- Cramton, C. (2001). The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science* , 346-371.
- Crowston, K., & Scozzi, B. (2003). Open source software projects as virtual organizations: Competency rallying for software development . *IEE Proceedings software* , 3-17.
- Crowston, K., Annabi, K., & Howison, J. (2003). Defining open source software project success. *Twenty-fourth international conference on information systems*, 2-14.
- Crowston, K., K, L., Wei, K., Eseryel, ., U., & Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, 564-575.
- Cukier, W., Ngwenyama, O., Bauer, R., & Middleton, C. (2009). A critical analysis of media discourse on information technology: preliminary results of a proposed method for critical discourse analysis. *Info Systems Journal*, 175-196.
- Currie, W., & Parikh, M. (2005). A Critical Analysis of IS Innovations Using Institutional Theory. *Journal of Management*, 317-356.

- Cusumano, M. A. (1991). *Japan's Software Factories* . New York: Oxford University Press.
- Cyert , R., & March, J. (1963). A behavioral theory of the firm . *University of Illinois at Urbana-Champaign's Academy for Entrepreneurial Leadership Historical Research Reference in Entrepreneurship* .
- Dahlander, L. (2005). Appropriation and appropriability in open source software . *international journal of innovation management* , 259-285.
- David, P. A. (2000). A tragedy of the public knowledge "comons"? *Stanford Institute for Economic Policy Research*.
- Dijksterhuis, A. B. (2006). On making the right choice: the deliberation-without-attention effect. *Science*, 1005-1007.
- Drake, B., & Moberg, D. (1986). Communicating influence attempts in dyads. *Academy of management* , 567-584.
- Drupal . (2011). *Drupal Core*. Retrieved 2013, from Drupal .
- Drupal . (2013). *Core Developers* . Retrieved 06 2013, from Drupal: <https://drupal.org/node/21778>
- Drupal . (2013). *Homepage*. Retrieved 2013, from Drupal: <https://drupal.org/home>
- Drupal. (2006). *Two New Core Committers* . Retrieved 2013, from Drupal: <https://drupal.org/node/51536>
- Drupal. (2013). *About Drupal*. Retrieved 2013, from Drupal.org: <http://drupal.org/about>
- Drupal. (2013). *Community - member directory* . Retrieved 2013, from Drupal : <https://drupal.org/profile>
- Drupal. (2013). *Contribute to development* . Retrieved 6 2013, from Drupal: <https://drupal.org/contribute/development>
- Drupal. (2013). *Core Developers* . Retrieved 2013, from Drupal.org: <http://drupal.org/node/21778>
- Drupal. (2013). *Download and Extend*. Retrieved 2013, from Drupal: <https://drupal.org/project/drupal>
- Drupal. (2013). *History*. Retrieved 2013, from Drupal.org: <http://drupal.org/about/history>
- Drupal. (2013). *Patches*. Retrieved 06 2013, from Drupal: <https://drupal.org/patch>
- Falbe, C., & Yukl, G. (1992). Consequences for Managers of Using Single Influence Tactics and Combinations of Tactics . *Academy of Management Journal*, 638-652.
- Fichman, R. (2004). Going beyond the dominant paradigm of information technology innovatoin research. *Journal for the association of information systems* , 314-355.
- Fultner, B. (2002). *On the Pragmatics of Social Interaction: Preliminary Studies in the Theory of Communicative Action*. MIT Press.

- Gallagher, S., & West, J. (2006). Patterns of Open Innovation in Open Source Software . In *Open Innovation: Researching a New Paradigm*. Oxford University Press.
- Gallivan, M. (2001). Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. *Information Systems Journal*, 277-304.
- Gallivan, M. J. (2001). Organizational adoption and assimilation of complex technological innovations: development and application of a new framework. *The DATA BASE for advances in information systems*, 51-86.
- Gattiker, T., & Carter, C. (2010). Understanding project champions' ability to gain intra-organizational commitment for enviromental projects. *journal of operations management*, 72-85.
- Gioia, D. (1986). Symbols, scripts, and sensemaking: Creating meaning in the organizational experience . In D. Gioia, *The Thinking Organization* (pp. 49-74). San Francisco: Jossey-Bass.
- Glass, R. L. (2006). *Software creativity 2.0*. Atlanta: Developer Books .
- Golant, B., & Sillince, J. (2007). The Constitution of Organizational Legitimacy: A Narrative Perspective. *Organization Studies*, 1149-1167.
- Guba, E. G. (1990). *The Paradigm Dialog*. NewBury Park: SAGE.
- Habermas, J. (1970). Towards a theory of communicative competence. *Inquiry*, 360-375.
- Habermas, J. (1974). *Theory and Practice* . Boston: Beacon Press.
- Habermas, J. (1975). *Legitimation crisis*. Boston: Beacon Press.
- Habermas, J. (1985). *The theory of Communicative Action: Volume 2: Lifeworld and Sytem: A Critique of Functionalist reason*. Beacon Press.
- Habermas, J. (1989). *The structural transformation of the public sphere* . Cambridge, MA: Massachusetts Institute of Technology Press.
- Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Elsevier Science*, 1159-1177.
- Innes, J., & Booher, D. (1999). Consensus building and comples adaptive systems. *journal of american planning association*, 412.
- Kaganer, E., Pawlowski, S., & Wiley-paton, S. (2010). Building Legitimacy for IT Innovations: The Case Physian Order Entry System. *Journal of the association of information systems* , 1-33.
- Kahneman, D., Knetsch , J. L., & Thaler, R. (1986). Fairness as a Constraint on Profit Seeking: Entitlements in the Market. *The American Economic Review*, 728-741.

- Kakabadse, A., & Parker, C. (1984). *Power, Politics and Organizations: A behavioural science view*. Chichester: Wiley and Sons.
- Kasper-fuehrera, E. A. (2001). Communicating trustworthiness and building trust in interorganizational virtual organizations. *journal of management*, 235-254.
- Khalil, O. W. (2002). Information technology enabled meta-management for virtual organizations. *international journal of production economics*, 127-134.
- Kipnis, D., Schmidt, S. M., & Wilkinson, I. (1980). Interorganizational Influence Tactics: Exploration in Getting One's Way. *Journal of Applied Psychology*, 440-452.
- Koch, S., & Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 27-42.
- Kogut, B., & Metiu, A. (2001). Open-Source Software Development and Distributed Innovation. *Open-Source Software Development and Distributed*, 248-264.
- Kotter, M. (2003). Negotiation of meaning and codeswitching in online tandems. *language learning and technology*, 145-172.
- Kramer, F. (1975). Policy analysis as ideology. *Public Administration Review*, 509-517.
- Krogh, G. v. (2003). Open-Source Software Development. *MIT Sloan Management Review*, 44(3).
- Lakhani, K. R., & von Hippel, E. (2003). How open source software works: "free" user-to-user assistance. *MIT Sloan School of Management*, 32(6).
- leiner, b., cerf, v., clark, d., kahn, r., kleinrock, l., lynch, d., et al. (1997). the past and the future history of the internet. *communications of the ACM*.
- Lerner, J. (1995). Patenting in the shadow of competitors . *Journal of Law and Economics* .
- Lerner, J., & Tirole, J. (2001). The open source movement: Key research questions. *European Economic Review*, 819-826.
- Ljungberg, J. (2000). Open source movements as a model for organising . *European Journal of Information Systems*, 208-216.
- Lyytinen, K. (1992). Information systems and critical theory . *Sage Publications*.
- Lyytinen, K. J., & Ngwenyama, O. K. (1992). What does computer support for cooperative work mean? a structural analysis of computer supported cooperative work. *Accounting management and information technology*, 19-37.
- Maciel, C. &. (2007). Design and metric of a 'democratic citizenship community in support of deliberative decision-making. *In Electronic Government*, 388-400.

- Malhotra, A., & Majchrzak, A. (2004). Enabling knowledge creation in far-flung teams: Best practices for IT and knowledge sharing . *Journal of knowledge management* , 75-88.
- Malone, T. Y. (1987). Electronic markets and electronic hierarchies. *Communication of the ACM*, 484-497.
- Martin, R. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Saddle River, USA: Prentice Hall.
- Marx, T. (1991). Removing the obstacles to effective strategic planning. *Long Range Planning*, 21-28.
- Mazzoleni , R., & Nelson, R. R. (1998). Economic Theories about the Benefits and Costs of Patents. *Journal of Economic Issues*, 32(4), 1031-1052.
- Mechanic, D. (1962). Sources of Power of Lower Participants in Complex Organizations. *Administrative Science Quarterly* , 349-364.
- Mockus, A., Fielding, R., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *22nd International Conference on Software Engineering*, (pp. 309-346). Limerick, Ireland.
- Mustonen, M. (2003). Copyleft-the economics of linux and other open source software. *Information Economics and Policy* , 99-121.
- Myers, M., & Klein, H. (2011). A set of principles for conducting critical research in information systems . *MIS Quarterly*, 17-36.
- Neilson, E., & Rao, M. (1987). The Strategy-Legitimacy Nexus: A Thick Description. *Academy of Management Review* , 523-533.
- Nelson , R. R., & Winter , S. G. (1982). *An Evolutionary Theory of Economic Change* . Harvard Collage .
- Ngwenyama, O. (1991). The critical social theory approach to information systems. *information systems research*.
- Ngwenyama, O. (1993). Developing end users' system development competence . *Information and Management* , 291-302.
- Ngwenyama, O. (1998). Groupware, social action and organizational emergence: on the process dynamics of computer mediated distributed work. *Accounting, Management, and Information Technology* , 127-146.
- Ngwenyama, O., & Lee , A. (1997). Communication Richness in Electronic Mail: Critical Social Theory and Contextuality of Meaning . *MIS Quarterly* , 145-167.
- Ngwenyama, O., & Lee, A. S. (1997). Communication Richness in Electronic Mail: Critical Social Theory and the Contextuality of Meaning. *MIS Quarterly*, 145-167.

- Ngwenyama, O., & Lyytinen, K. J. (1997). Groupware technologies as action constitutive resources: A social action framework for analyzing groupware technologies . *The journal of collaborative computing* , 71-93.
- Ngwenyama, O., & Nielson, P. (2013). Using organizational influence processes to overcome IS implementation barriers: Lessons from a longitudinal case study of SPI implementation . *European journal of information systems*.
- Oliver, C. (1991). Strategic Responses to Institutional Processes. *Academy of Management Review*, 145-179.
- Olle Persson, W. G. (2004). Inflationary Bibliometric Value: the role of scientific collaboration and the need for relative indicators in evaluative studies. *Scientometrics*, 421-432.
- Olson , G., & Olson, J. (2000). Distance Matters . *Human-Computer Interaction* , 139-178.
- O'Mahoney, W. &. (2008). The role of participation architecture in growing sponsored open source communities . *Industry and Innovation* .
- O'Mahony, S. &. (2007). The emergence of governance in an open source community . *Academy of Management journal* .
- O'Mahony, S. (2007). The governance of open source initiativesL what does it mean to be community managed? *Journal of Management and Governance*, 139-150.
- Osterloh, M., & Rota, S. (2007). Open source software development-just another case of collective invention . *research policy* , 157-171.
- Pattigrew, A. (1985). *The awakening giant: Continuity and change in Imperial Chemical Industries*. New York, NY: Blackwell.
- Pfeffer, J. (1981). Management as Symbolic Action: The Creation and Maintenance of Organizational Paradigms . *Research in organizational behaviour* , 1-52.
- Polanyi, M. (2002). Communicative action in practice: future search and the pursuit of an open, critical and non-coercive large-group process. *system research and behavioral scienc*, 357-366.
- Pondy, L. (1978). Leadership is a language game. In M. McCall, & M. lombardo, *Leadership where else can we go* (pp. 88-99). Durham, NC: Duke University Press.
- Roberts, E. (2007). Managing invention and innovation . *Research Technology Management* , 35-54.
- Salancik, G., & Meindl, J. (1984). Corporate attribution: Strategic illusions of management control. *adminstrative science quarterly*, 238-254.
- Schriesheim, C., & Hinkin, T. (1990). Influence tactics used by subordinates. *American Psychological Association*, 246-257.
- Schumpeter, J. (1934). *The Theory of Economic Development*. Harvard University Press.

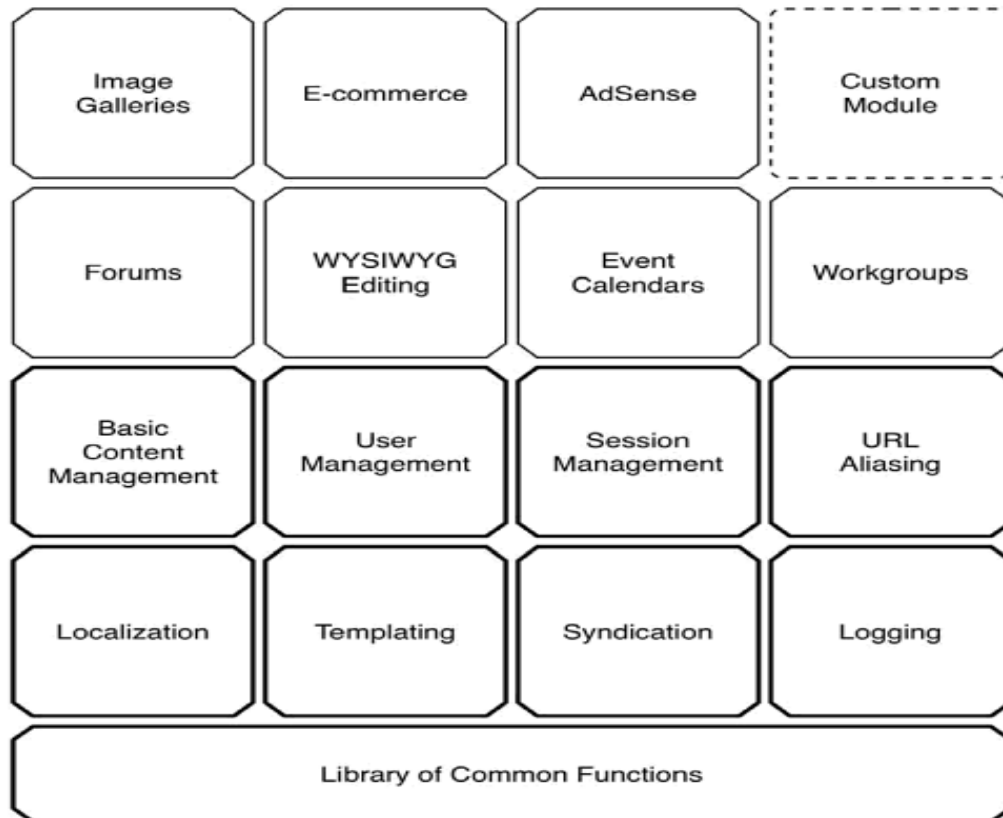
- Scott, W. (2001). *Institutions and Organizations* . Thousand Oaks: Sage Publications.
- Simons, T. P. (1999). Making use of difference: Diversity, debate, and decision comprehensiveness in top management teams. *Academy of management journal*, 662-673.
- Soderberg, A. K. (2013). Global software development: Commitment, Trust and Cultural sensitivity in strategic partnerships. *Journal of international management*.
- Stewart, K. J., & Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly*, 291-314.
- Strang, D., & Macy, M. (2001). In Search of Excellence: Fads, Success Stories, and Adaptive . *American journal of sociology* , 147-182.
- Strang, D., & Soule, S. (1998). Diffusion in organizations and social movements: From Hybrid. *Annual review of sociology* , 265.
- Subramanyam, R., & Krishnan, M. (2003). Empirical analysis of CK metrics for object oriented design complexity: Implication for software defects . *IEEE Transactions on software engineering*, 297-310.
- Suchman, M. (1995). Managing Legitimacy: Strategic and Institutional Approaches. *The academy of management review*, 571-610.
- Sussman, M., & Vecchio, R. (1997). *A social influence interpretation of worker motivation*. Univeristy of Notre Dame Press.
- Swanson, E., & Ramiller, N. (1997). The organizing vision in information system innovation . *Organization science*, 458-474.
- Tatikonda, M. R. (2000). Technology novelty, project complexity, and product development project execution success: a deeper look at task uncertainty in product innovation. *IEEE transactions on Engineering Management*, 74-87.
- Terttu Luukkonen, O. P. (1992). Understanding Patterns of International Scientific Collaboration . *Science, Technology and Human Values* .
- Tirole, J., & Lerner, J. (2002). Some simple economics of open source . *The Journal of Industrial Economics* , 197-234.
- Tjosvold, D., Tang, M., & West, M. (2004). Reflexivity for team innovation in china. *Group Organization Management*, 540-559.
- Vainio, N., & Vaden , T. (2007). *Free Software Philosophy and Open Source*. Hershey, PA: Information Science Reference .
- Van Der, H., & Schoemaker, P. (1992). *Integrating scenarios into strategic planning at royal dutch/shell*. MCB UP.

- VanDyk, J. K., & Westgate, M. (2007). *Pro Drupal Development*. New York: Springer-Verlag.
- von Hippel, E., & von Krogh, G. (2003). Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, 209-223.
- Waterman, M. J. (2013). the effect of complexity and value on architecture planning in agile software development. *Engineering and extreme programming*, 238-252.
- Wylant, B. (2008). Desinging thinking and the experience of innovation. *Design Issues*, 3-14.
- Ye, Y. K. (2003). Towards an understanding of the motivation of open source software developers. *Software Engineering*, 419-429.
- Yukl, G., & Falbe, C. M. (1990). Influence Tactics and Objectives in Upward, Downward, and Lateral Influence Attempts. *Journal of Applied Psychology*, 132-140.
- Yukl, G., & Tracey, B. (1992). Consquences of influence tactics used with subordinates, peers, and the boss. *American Psychological Association*, 525-535.
- Yukl, G., Falbe, C., & Youn, J. (1993). Patterns of Influence Behavior for Managers. *Group Organization Management*, 5-28.
- Zeitlyn, D. (2003). Gift economies in the development of open source software: anthropological reflections. *Research policy*, 1287-1291.

APPENDCES

APPENDIX A

The figure below illustrates the Drupal core software and modular design. The bold boxes represent the core functions, while the rest of the boxes are added modules.



(VanDyk & Westgate, 2007)

APPENDIX B

The table below illustrated examples of instrumental action in the Drupal Deliberations.

#	Data Type	Case	Empirical Evidence (Quote)	Explanation
1	Instrumental Action – Community Feedback	5	January 10, 2005 - 20:29 : Dries Can someone review this please?	Dries wants further feedback and code review from the community in order to Commit this patch.
2	Instrumental Action – Community Feedback	12	November 23, 2004 - 23:24 : Dries I'd like to move forward with this patch and include it in Drupal 4.5.1. I can't reproduce this problem (it seems) so it would be much appreciated if those who can, can test it.	Dries wants further feedback and code review from the community in order to Commit this patch.
3	Instrumental Action – Community Feedback	75	February 10, 2005 - 15:02 : Dries I'll commit this patch when there is more demand for it.	Dries is seeking more interest for this patch before he commits it.
4	Instrumental Action – Technical Input	12	December 15, 2004 - 21:33 : Dries If possible provide a single patch against DRUPAL-4-5 and a second patch against HEAD. Looks like the patches are no longer in sync.	Dries is requesting a technical modification to the patch.
5	Instrumental Action – Technical Input	31	Because of a bug in the project module, you can't use '.inc' in the name of your patch. Please rename and resubmit your patch. Dries	Dries requesting a technical modification to the patch.

APPENDIX C

The table below shows examples of comments illustrating low complexity innovations and high complexity innovations in the fast commit category. The purpose of this table is to illustrate the criteria that differentiate low and high complexity innovations in this category.

Data Type	Case	Empirical Evidence (Quote)	Criteria of complexity	
Low Complexity Innovation	Thread 74	February 8, 2005 - 01:10 : drumm Attachment: http://drupal.org/files/issues/page.module_2.diff (1.56 KB) The page module's long help text is a bunch of lies and then it briefly explains it's permissions. IMO it should just be taken out. I can't think of what help should be there.	Difficulty	Simple, writing text
			Steps required:	One step
			Interaction with rest of software:	none
Low Complexity Innovation	Thread 81	March 9, 2005 - 14:54 : Morbus Iff The following patches clear up a number of minor inconsistencies during a Drupal installation. Largely, this is related to internal documentation and "where things go", but should help organize, cleanup, and ease the installation. These aren't features, so they're eligible for a 4.6 commit.	Difficulty:	Addressed minor inconsistencies, mostly text.
			Steps required:	One step
			Interaction with rest of software:	None
Low Complexity Innovation	Thread 86	March 25, 2005 - 20:41 : JonBob Attachment: http://drupal.org/files/issues/descriptions.patch (18.88 KB) We have tons of different ways of phrasing module descriptions for the module listing page. This patch rephrases them all to consistently use an active verb, and cleans up some grammar and clarity issues.	Difficulty:	Simply rephrases module description using active verbs.
			Steps required:	One step
			Interaction with rest of software:	None
Low Complexity Innovation	Thread 85	March 28, 2005 - 14:24 : Uwe Hermann Attachment: http://drupal.org/files/issues/INSTALL.txt.patch (1.44 KB)	Difficulty:	Tiny patch
			Steps required:	One step

		Hi, here's a tiny patch which fixes some issues in the INSTALL.txt file.	Interaction with rest of software:	None
Low Complexity Innovation	Thread 84	March 31, 2005 - 06:48 : Uwe Hermann Attachment: http://drupal.org/files/issues/spellcheck_0.patch (9.1 KB) I ran ispell over the whole Drupal code (including themes etc.). Here's a patch with the fixes I (or ispell) found.	Difficulty:	Simply a spell check
			Steps required:	One step
			Interaction with rest of software:	None
High Complexity Innovation	Thread 57	November 19, 2004 - 16:03 : jasper 1. "my account" link can not be given "weight" 2. If edited, "my account" link behaves strangely: appears as separate menu in admin screen, disappears from navigation menu, etc. 3. Subitems can't be added under my account, or, strange things happen if you try to do that.	Difficulty	More than one issue, and requires trouble- shooting.
			Steps required:	multi-step solution
			Interaction with rest of software:	Many
High Complexity Innovation	Thread 76	January 26, 2005 - 14:18 : moshe weitzman Attachment: http://drupal.org/files/issues/drdest.patch (11.05 KB) Here is a patch I've been wanting to finish for a while. This patch assures that you end up on the proper page after you edit/delete a node, comment, user, or url alias. This is true no matter if you go through the usual interface or the admin interface. Further, if click the 'edit' link from 3rd page of a custom sorted view (e.g. admin/comment&from=100&sort=asc&order=Author) you still are returned to the right page.	Difficulty	Intricate
			Steps required:	Multi step solution
			Interaction with rest of software:	Many
High Complexity Innovation	Thread 83	Attachment: http://drupal.org/files/issues/multi_db_connections.diff (5.23 KB) This patch allows multiple database connections to be used within Drupal. The method of specifying :	Difficulty	This an innovation that allows drupal to connect to multiple databases,

		<pre><?php \$db_url = "mysql://user:pass@localhost/dbname"; ?></pre> <p>will still work, it will create the 'default' connection. However, if you need multiple connections.. you can specify :</p> <pre><?php \$db_url["default"]?????= "mysql://user:pass@localhost/dbname"; \$db_url["other_db"]?????= "mysql://user:pass@localhost/other_dbname"; \$db_url["other_server"] = "mysql://user:pass@my.server.com/dbname"; ?></pre>	Steps required:	Multi-step solution
			Interaction with rest of software:	More than one

APPENDIX D

The table below shows examples of comments illustrating low complexity innovations and high complexity innovations in the slow commit category. The purpose of this table is to illustrate the criteria that differentiate low and high complexity innovations in this category.

Data Type	Case	Empirical Evidence (Quote)	Criteria of complexity	
High Complexity Innovation	Thread 14	<p>December 30, 2004 - 16:41 : Morbus Iff</p> <p>I recently upgraded from 4.4.2 to 4.5.1, and have noticed that the poll block no longer displays the "add new comment" or "# comments" link. This is (was) important to me, as I usually ask for comments whenever someone chooses the negative option. Any idea where and why this went?</p> <p>You can still get to the comment form in a roundabout way ("other polls", choose your poll, whammo), but that's far too many clicks and intellect points.</p>	Difficulty	Requires troubleshooting
			Steps required:	Multi-step
			Interaction with rest of software:	none
High Complexity Innovation	Thread 3	<p>January 12, 2005 - 11:02 : Morbus Iff</p> <p>The automated unpublishing is great for stopping spam from being displayed, but it doesn't stop spam from corrupting another feature: my "recent posts" (tracker.module). I'll regularly get bursts of 200 auto-unpublished spams which make the "recent posts" page useless – an unpublished spam still affects the modification date of a node, which makes "recent posts" display craploads of updates, even though there really isn't any. Thus, the feature request would be:</p> <ul style="list-style-type: none"> - treat an unpublished spam as a deleted spam - when a spam is automatically unpublished, revert the node modification date to its previous value. <p>I'm not entirely sure how easily possible this is, as you'd have to make sure the date is properly set when someone says "nah, this unpublished spam was actual ham", and then republishes it.</p>	Difficulty:	Problem identified, but intricate
			Steps required:	Multi-step
			Interaction with rest of software:	few

High Complexity Innovation	Thread 77	February 14, 2005 - 08:51 : tangent	Difficulty:	Problem identified
		Attachment: http://drupal.org/files/issues/settings-session.patch (1.71 KB)	Steps required:	Multi-step
		<p>As discussed in this issue [1], it would be desirable to move the session settings into /sites/default/settings.php so that subsites can have better control over them. One of the advantages of the site specific settings.php file is that it will never get overwritten during upgrades and having these settings here should prove to be more friendly. I have created a patch which moves most of the PHP session settings from .htaccess to /sites/default/settings.php with the exception of "session.auto_start" because it must not, as far as I know, be modified.</p> <p>I have also added 2 additional commented settings which I suspect are often needed as they were in my case.</p> <p>[1] http://drupal.org/node/2974</p>	Interaction with rest of software:	Many, requires moving functions
Low Complexity Innovation	Thread 85	October 21, 2004 - 20:42 : drumm	Difficulty:	Problem identified, but solution is complex
		Attachment: http://drupal.org/files/issues/tmp (42.68 KB)	Steps required:	Multi- step
		<p>The primary goal of this patch is to take the 'custom' and 'path' columns of the block overview page and make them into something understandable. As of Drupal 4.5 'custom' lacked an explanation which wasn't buried in help text and path required dealing with regular expressions.</p> <p>Every block now has a configuration page to control these options. This gives more space to make form controls which do not require a lengthy explanation. This page also gives modules a chance to put their block configuration options in a place that makes sense using new operations in the block hook.</p> <p>The only required changes to modules implementing hook_block() is to be careful about what is returned. Do not return anything if</p>	Interaction with rest of software:	None

		<p>\$op is not 'list' or 'view'. Once this change is made, modules will still be compatible with Drupal 4.5. Required changes to core modules are included in this path.</p> <p>An additional optional change to modules is to implement the additional \$op options added. 'configure' should return a string containing the configuration form for the block with the appropriate \$delta. 'configure save' will come with an additional \$edit argument, which will contain the submitted form data for saving. These changes to core modules are also included in this patch.</p> <p>I have posted screenshots at http://foo.delocalizedham.com/tmp/blocks/. Additional work, such as further rearrangement of the block overview page, is as always a task for another patch.</p>		
Low Complexity Innovation	Thread 75	<p>February 1, 2005 - 21:26 : kbahey</p> <p>Attachment: http://drupal.org/files/issues/contact.module-subject.patch (1.93 KB)</p> <p>I find it very undescriptive when I receive a message from Drupal with the subject "message from username". This patch adds a "subject" field for the contact.module which the user can fill, and would tell you what they want right away. Oh, and it helps group the 'conversation' on Gmail into something meaningful. (Note, I have not tested this since I do not have a CVS installation at the moment. Appreciate if someone can test it).</p>	Difficulty:	Problem identified, application is simple
			Steps required:	One step
			Interaction with rest of software:	None

APPENDIX E

This table provides a summary of all characteristics for all 24 Committed Cases. This is to be used for a high level view of the difference between the characteristics of fast and slow commits.

Characteristics	Fast Commits					Slow Commits				
Number of Cases	16					8				
Complexity of Innovation	High Complexity	5				High Complexity	7			
	Low Complexity	11				Low Complexity	1			
Average Number of Actors in Conversations	3.7 Actors					5.9 Actors				
Range of Actors in Conversations	2 to 6					4 to 9				
Number of Influencing Contributors in All Cases	1 Influencer	12 Cases				1 Influencer	3 Cases			
	2 Influencers	3 Cases				2 Influencers	2 Cases			
	3 Influencers	1 Cases				3 Influencers	1 Cases			
						4 Influencers	1 Cases			
						5 Influencers	1 Cases			
Average Number of Comments in Conversations	5.3 Comments Per Case					17 Comments Per Case				
Range of Number of Comments	2 to 8 Comments					9 to 40 Comments				
Number of Communicative Action Types	Action Type	Total	Min	Max	Mean	Action Type	Total	Min	Max	Mean
	Communicative	75	2	7	4.7	Communicative	112	7	39	10.4
	Discursive	8	0	3	0.5	Discursive	18	0	6	2.3
	Instrumental	2	0	1	0.13	Instrumental	6	0	4	0.8
Request for Input by Dries	2 Requests by Dries					7 Requests by Dries				
Influence Tactics Used	Rational Persuasion	32				Rational Persuasion	52			
	Consultation	6				Consultation	11			
Influence Tactics by Conversation Initiators	Rational Persuasion	29				Rational Persuasion	28			
	Consultation	3				Consultation	3			
Validity Claims Challenged	Comprehensibility	2				Comprehensibility	1			
	Effectiveness	3				Effectiveness	3			
	Efficiency	5				Efficiency	13			
	Relevance	1				Relevance	1			
Type of Innovation	Bug Report	4				Bug Report	5			
	Feature Request	5				Feature Request	2			
	Task	4				Task	1			

APPENDIX F

Reflection

In the summer of 2010 I was introduced to a thesis project that would examine the effects of Software Process Improvement (SPI) on the quality of globally distributed software. I wrote a proposal for this project, and frequently discussed with my thesis advisor the literature review, method of data collection and interrogation, and theoretical frameworks to consider using. Months later, at the same time that a proposal was developed and the study was about to enter the data collection stage, I arrived at a crossroads. My thesis advisor had spoken to me about a potential project that he was planning; it was to do with the innovations that develop within virtual software production methodologies, open source software to be specific. The study would examine the underlying dynamics of the legitimation processes of these innovations. I became very interested in that project and found its potential more interesting to me than that of the initial study, and after consulting with my thesis advisor I decided to abandon the initial study and commence on to the open source field. It was not an easy decision due to the time already invested in the initial study, but it seemed worth it when I thought about the yet to come investment of time and effort into the thesis, I thought that the more interesting topic will allow me to more effectively assimilate the concepts of the study. At that point I had to make a choice between building a questionnaire to collect data for my first study, or embark on this journey and decipher the content of the raw text file of the new study, I decided that I rather the latter. Within 3 months, I had written a literature review, and had skimmed through the data file several times, with the aim of better understanding the nature of the conversations. Although it was intimidating to look at the text and think that not only will it be understood, but also analyzed beyond the explicit meanings, a practice that I had no experience in yet, I felt more related to the new project. I believe that due to my familiarity with online discussion threads on automobile and technology forums, I was able to assimilate the concepts of the study effectively. I felt that once I identify the beginning and end of each conversation, and the subject matter, the data will analysis process will be feasible. Of course there was a learning curve once it was time to apply the text against the Communicative Action and Influence Tactics theories, but after few iterations of coding it felt familiar.

It was during that point of my thesis that I understood my supervisor's advice to try and relate to the text as much as possible, and I realized that understanding the data on a personal level will provide with a smoother journey to assimilating and going beyond the explicit. I learnt that empirical data takes the lion's share of effort in the thesis, specifically the coding phase, and to be able to best predict one's level of comfort with a study, one must first either examine the data at hand, or understand the nature of the data that is to be collected, to gauge the level of familiarity with data. The more personally related and familiar a researcher feels towards the data, the smoother the journey through the different phases of the study.

Also, another lesson I learnt from my experience in this project is that although one might be tempted to study an interesting phenomenon, one must be careful not to include a bigger chunk of a phenomena to study in an allocated time frame than one can afford because it may turn out that the scope of the study is too large to complete in an allocated expense or time frame. Unlike I knew when this project started; studying what could appear to be a simple social phenomenon could end up being not so simple.

What I Learnt About the Field of Social Science

Firstly, I learnt that writing a thesis is not only about reporting findings and building a solid case, but also about building an aesthetically pleasing thesis that will be less likely to intimidate readers. I learnt this the hard way when my thesis advisor thought that my 'Discussion of Empirical Analysis' chapter contained too many figure, tables and numbers and not enough texts explaining those numbers. I restructured that chapter two times before I arrived to this version.

The Data Analysis

My thesis advisor was able to get his hands on text files that contained thousands of pages of conversations between members of the Drupal community. The key words innovation, open source and software production sparked my interest when I heard them. As I started to explore the data and consult with my thesis advisor, it became apparent to me that I will eventually need to gain a deep understanding of this raw text file that seemed many stages away from being in a format that is ready for analysis, and although I acknowledged that it will take several stages of coding to arrive at such stage, I did not anticipate the knowledge that I would gather during coding cycles, and the effect that this knowledge would have on proceeding coding cycles. I

had initially spoken to Dr. Ngwenyama about the Communicative Action and the Influence Tactic theories as appropriate aspects to interrogate the text; I anticipated that I would probably conduct two cycles of coding, one for each framework and perhaps an additional cycle to ensure the coherency of the coding. What followed was probably the part of the thesis that was most challenging for me to get through, namely due to the unstructured and raw format of the original text file, which was almost illegible during the first few times of viewing

I am still unsure what format the data was exported into the text file from, but it is most probable that the contents of the file were copied from email conversation threads, and pasted into a notepad format. Also, I had underestimated the amount of time it would take before I could understand the social context of Drupal in order to appropriately code the conversations, to the point that midway through the first round of coding I was still calibrating my judgement on how to apply codes from the aspects of the frameworks onto the text and had to run through a second cycle to make sure that the first portion of the coding was consistent with the rest.

In addition, as the coding for the first two aspects of the frameworks were done and I started to get a clearer picture of the dynamics of the conversation and the legitimization processes, I decided that I would need to study the explicit validity claims that were challenged to be able to explain different scenarios. This added an extra round of coding to the data analysis.

The Literature Review

The key lesson I learnt about writing a literature review for my thesis was that material within a discourse becomes easier to assimilate as one works on conceptualizing latter parts of thesis. The literature review does not need to be completed before the rest of the thesis; in fact I would frequently come back to the literature review to add components.

In my case this also proved to be the case for the Research Methodology chapter, which became easier for me to explain as the thesis progressed. It is very important however to write the steps that the data collection and analyses went through, because the details of the research processes could be easily forgotten.