The Distributed Nature of Software Development – a comparison of three development approaches

Helena Holmström

Viktoria Institute
Box 620, SE-405 30 Gothenburg, Sweden
helena.holmstrom@viktoria.se

Abstract

Much change has undergone the environment in which software development takes place. To a greater extent, we are experiencing a distributed development environment. In this paper, three approaches to distributed software development are identified and explored – global software development (GSD), open source software development (OSS) and community-based software development (CSD). In a comparison of these, it is argued that the approaches embrace differences that are important to take into consideration for companies entering the distributed environment of software development. This paper suggests that these differences are related to the dimensions of (1) nature of development approach, (2) communication structure, and, (3) coordination mechanisms.

Keywords

Distributed software development, distributed development approaches

1 Introduction

Much change has undergone the environment in which software development takes place (Sheremata, 2002; Orlikowski, 2002; Feller & Fitzgerald, 2002). Today, software developers work under increasing competitive pressures and the systems themselves are continuing to be more technically advanced. Also, there are changing customer requirements to take into consideration and an accelerating demand for more customizable features in the software that is produced. There is little doubt that these aspects make software development an unwieldy process.

However, what really strike you are the changes in the way the software development process is coordinated. To a greater extent, software development is becoming a distributed process and there is the need for development approaches that take into consideration the distributed environment in which developers and users communicate and coordinate their work.

In this paper, three approaches to distributed software development are identified and explored. First, there is global software development (Carmel & Agarwal, 2001; Dubé & Paré, 2001; McDonough et al, 2001). Second, there is open source software development (Feller & Fitzgerald, 2002; Raymond, 1999; Gallivan, 2001; Sharma et al, 2002; Bergquist & Ljungberg, 2001). Third, there is community-based software development (Holmström, 2001).
To illustrate the different approaches, three empirical cases are presented. In a comparison of these, it is argued that the approaches embrace differences in terms of: (1) nature of development approach, (2) communication structure, and, (3) coordination mechanisms. To this end, this paper aims at creating an understanding of the different approaches to distributed software development that can be taken, and in this way, facilitate for software development companies entering the distributed development environment.

2 Background

Below, three approaches to distributed software development are identified – global software development (GSD), open source software development (OSS) and community-based software development (CSD). In the following discussion, these different approaches constitute the background for a comparison of three empirical cases.

2.1 Global Software Development

In global software development (GSD), geographically distributed, and culturally diverse, software developers or development teams work jointly in a software development project (McDonough et al, 2001). According to Carmel and Agarwal (2001), there are at least 50 nations participating in collaborative software development internationally, and the number is rapidly increasing due to economical benefits of outsourcing, the growth of the global market and the occurrence of business arrangements such as strategic partnerships and joint ventures (Karolak, 1998). To support this type of distributed development there is the need for communication and cooperation technologies. As recognized by Smith and Blanck (2002), a combination of technologies can be used where synchronous media such as videoconference systems, electronic meeting systems and virtual whiteboards are combined with asynchronous media such as voicemail, electronic bulletin boards and forums, e-mail and group calendars.

Global software development is not characterized of user-driven development. Instead, the software is developed by professional developers and then sold to users around the world. In this sense, the users can be seen as consumers of the value that is produced. Hence, the key challenge during software development is not on techniques for user-developer interaction, but rather on techniques to support developer-developer communication and coordination.

2.2 Open Source Software Development

In conformity with global software development in which software development is performed by geographically distributed developers, there is open source software (OSS) development. Recently, this movement has gained significant attention and it is believed that OSS development has the potential to influence the future of organizations both in terms of organization, customer relations and business models (Ljungberg, 2000).

In OSS, communities of developers contribute on a voluntary basis in developing software that is freely shared for review, reuse and modification. As recognized by Ljungberg (2000), the work seems to be totally distributed, delegated and loosely coupled. In terms of organization, the OSS development approach does not have any formal structure (Sharma et al, 2002). Hence, projects are not dictated by any formal schedule or list of deliverables, neither is work assigned to the developers. No particular development method is advocated and unlike conventional software development there is no formal procedure to ensure that...
developers are not duplicating effort by working on the same problem at the same time. On the contrary, this is seen as beneficial to the process since it allows for a competition among multiple high-quality solutions (Feller & Fitzgerald, 2002). To coordinate the process, configuration tools such as the Concurrent Versions System (CVS) are used. The CVS offers an easy way to incorporate changes to the repository and with one single command the developers can download the latest version of the software tree (Feller & Fitzgerald, 2002). Lately, the CVS has also been complemented with web-based extensions such as Bugzilla (web-based bug tracking), Bonsai (web-based access to archived source code) and Tinderbox (web-based tools for analysing software builds).

In contrast to global software development, OSS development is characterized by a complex definition of who is the user and who is the developer. Often, the developer and the user of one particular piece of code is the same person. According to Ljungberg (2000), this is due to the fact that most OSS projects originate in individual needs and requirements.

2.3 Community-based Software Development

Recently, there have been studies highlighting the importance of virtual communities as platforms for software development (Holmström, 2001; Henfridsson and Holmström, 2002). In particular, OSS literature has contributed to the view of virtual communities as enablers for collaborative work between distributed people (Sharma et al, 2002; Scacchi, 2002). As recognized by Sharma et al (2002), OSS development is a fundamentally new way to develop software and the large number of voluntary developers reflects the strength of the community culture and the ‘sense of community’ that can be found in virtual groups of like-minded people (Blanchard and Markus, 2002).

However, while there are significant benefits of OSS development, there is not always the possibility for software companies to deploy this way of distributed development. Most often, for-profit organizations have difficulties in building business models around the OSS paradigm (Sharma et al, 2002). Instead, there is the challenge of finding ways to incorporate aspects of the community culture into traditional software development processes – and in this way to allow for a community-based approach to software development in which user involvement and user participation are key characteristics.

To do this, there is the possibility to create ‘hybrid communities’ (Sharma et al, 2002). In these, features found in OSS communities are infused to varying degrees into traditional organizational structures to facilitate for flexible and user-driven development of quality software. Evidence points to leading organizations like Hewlett Packard, IBM, Intel, Sun Microsystems, etc., already having taken steps to use communities as a way to incorporate elements of OSS into their software development processes (Sharma et al, 2002). This indicates the belief in communities as valuable for involving users in the development process. Also, the benefits of (a) reduced development time, (b) improved quality, (c) reduced cost, (d) gained developer loyalty, and (e) increased developer talent pool can be enjoyed (Sharma et al, 2002).

In resemblance with the hybrid communities presented by Sharma et al (2002), community-based software development requires the consideration of three major elements: (a) community building, (b) community governance, and (c) community infrastructure. First, ‘community building’ refers to the precondition of having a ‘community of practice’ (Wenger, 1998) with a strong and shared culture. To support this, organizations need to provide a free flow of information, to get rid of the formal organizational structures and
provide mechanisms for informal relationships and networking among community members (Sharma et al, 2002). Second, ‘community governance’ refers to the implementation of transparent governance mechanisms. Here, managers have to move away from the practice of imposing central command and control and, instead allow for community members to work in teams and to make decisions by discussing and voting (Sharma et al, 2002). Third, ‘community infrastructure’ refers to the tools and infrastructures necessary for software development. In resemblance with the CVS-system that is used within OSS development (Feller and Fitzgerald, 2002), community-based software development presupposes a central repository in which information is accessible for the community members (Sharma et al, 2002).

Based on this background, three empirical cases of distributed software development are presented. In the following discussion, the three cases are compared in terms of (1) nature of development approach, (2) communication structure, and (3) coordination mechanisms.

3 Research Setting and Method

The empirical part of this paper is based on three case studies. To obtain this data, two different methods for data collection were used. First, a secondary analysis of two published case studies on distributed software development was employed. Second, an interpretive case study at a software company was conducted.

First, the secondary analysis implied the identification of representative papers. To identify the first case study – the case on global software development – I used the search engine ‘Google’ and the scientific literature digital library ‘CiteSeer’ which uses the search engines AltaVista, HotBot, and Excite to identify publications within the field of computer science and information systems. In this search, I used terms such as ‘global software development’, ‘distributed software development’ and ‘distributed development teams’ to identify studies within this area. Although many publications were available on global software development, there were few case studies in which original data and data analysis could be obtained. Finally, by backtracking a reference found in a paper on product development, a study by Orlikowski (2002) was identified. In being an empirical account of the work conducted in a geographically-dispersed organization this study matched the search criteria and was selected to represent the global software development approach.

The second case study on which secondary analysis was employed was identified in the reading of *Information Systems Journal* and the special issue on *Open Source* (ISJ, nr 11, 2001). In one of the papers, Gallivan (2001) identifies nine case studies of OSS development. According to Gallivan, these studies were identified after searching the electronic archives of both ACM (The Association for Computing Machinery) and IEEE (Institute of Electronics and Electrical Engineers), after searching the database Bell & Howell/Proquest’s AB/Inform and after reviewing the papers that were presented at the 1st Workshop on OSS Engineering (Feller et al, 2001). From the hundreds of publications that were found, only nine fit the selection criteria outlined by Gallivan (2001). First, the publication needed to describe the process of OSS development, in general, or one or more specific OSS projects. Secondly, the paper had to contain original data and analysis. From the nine case studies that were identified by Gallivan (2001), the Moon and Sproull (2000) study was selected to represent the OSS development approach described in this paper.
The third case study presented in this paper is the study of community-based software development. The empirical work reported here builds on an interpretive case study (Walsham, 1995; Klein & Myers, 1999), conducted by me and a research colleague at Daydream Software between January 2000-October 2002 (for publications on this study see for example, Holmström, 2001; Nyberg and Henfridsson, 2001; Henfridsson and Holmström, 2002). In our study, we focused on the software development process of the online game Clusterball and the way in which a virtual community was used to involve distributed users in the development process. In the study, data sources such as technical documents, meeting protocols, press releases and printouts from the community forum were used. Also, an extensive review of other gaming websites was conducted. Furthermore, the specific context of Daydream and its customers was explored through 600 hours of participant observations at the company, 14 qualitative interviews with Daydream employees and a web-survey that was sent out to 200 community members.

4 Distributed Software Development: Three Empirical Cases

In section two, three approaches to distributed software development were identified. Below, these approaches are explored in three empirical cases. First, there is the Kappa case, representing global software development (Orlikowski, 2002). Second, there is the Linux case, representing OSS development (Moon and Sproull, 2000). Finally, there is the Clusterball case, representing community-based software development (Holmström, 2001). While the first two cases are based on secondary analysis of published case studies, the third case is based on an interpretive case study conducted by the author.

4.1 Global Software Development – the Kappa Case

Kappa is a globally-dispersed software development company with its headquarters in The Netherlands. At Kappa, the software development efforts are accomplished through temporary, global project groups involving a few hundred software engineers from all over the world. The software development activities are distributed across multiple Development Units (DU’s) located in 15 different locations spread over 5 continents. The geographical and cultural diversity of Kappa can be understood in the following statement made by one of the managers:

“My situation is quite typical...I am a Greek working in Finland for a Dutch company and using English to do my work.”

The rationale for having these distributed development teams is both economical and strategical. While the economical aspects include an increasing competitive pressure to reduce the time-to-market and an accelerating demand for more customizable features in the software, the strategical aspects are touched upon by one of the senior executives:

“...First of all, you get access to resources wherever it is. Holland is a pretty small country and our universities just don’t turn out the number of engineers that Kappa needs...Another advantage is proximity to the markets.”

To manage the complexity of the distributed work at Kappa there is a well-established organizational structure. In each location there are senior executives, senior DU managers,
project managers, subproject managers and software engineers. However, as observed by one of the senior executives, there are difficulties in being such a diverse organization:

“... as much as it is very nice to have these organizations that are diverse, they also sometimes pull in different directions. And the big challenge is to bring them together.”

To handle negotiations and discussions there are substantial aligning efforts within Kappa. These are accomplished through two key activities: (1) the use of a propriety project management model, its planning tool and structured systems development methodology, and (2) the annual contracting for work via standard metrics. A senior executive commented on the role of project planning and methodology tools in facilitating distributed work:

“We use a common process methodology...And then we have coordination within this framework, done at all levels of the project to get all the different software pieces together for the system at the same time. There are the technical standards and coordination documents....”

As part of the project plan, several documents have to be written. For example, there is an operational plan, an assignment specification and a project specification. The importance of Kappa’s project management model and methodology in aligning the different projects is explained by one of the project managers:

“The project model and methodology helps a lot...We develop requirement specifications, development sketches, implementation proposals, technical reports, everything that tells us at an early stage, this is the scope, this is feasible, this is what we are going to do and this is what it costs now.”

The use of the project management model as well as the division of projects into subprojects is all part of the hierarchical decomposition of work that characterizes Kappa. However, despite its efficiency of coordinating the different projects, the propriety suite of the project management model can be viewed as constraining in shifting to new software platforms, new infrastructures, new programming languages and new development methodologies. One of the managers commented about Kappa’s current project management model:

“I think it helps us, but the drawback is that the limit has been hit now of the capacity of that model...what we need now is a new model and a new methodology for parallel development.”

Also, Kappa members emphasize the importance of face-to-face meetings. Despite the qualities of the project management model, these are necessary for establishing social relationships. One project manager noted:

“You can’t resolve everything over the phone. It is important to have that personal relationship as well, which you achieve by meeting each other, and then it makes it a lot easier when you communicate through e-mail or the phone.”

The statements above reflect both advantages as well as limitations in working in a distributed environment. At Kappa, 30 nationalities in 15 geographic locations are currently trying to align their software development efforts. Their customers, on the other hand, seldom experience anything but a final product. Unaware of the distributed nature of Kappa they turn to their local subsidiary when having any problem.
4.2 Open Source Software Development – the Linux Case

In the beginning, Linux was a PC-based operating system produced through a software development effort consisting of more than 3,000 developers distributed over 90 countries on five continents. In its first three and a half years of development more than 15,000 people submitted code or comments to the three main Linux related newsgroups and mailing lists. As of December 1998, more than eight million users were running Linux on a wide variety of platforms and the operating system was projected to have an annual growth rate of 25% (Shankland, 1998). Today, Linux is much more than an operating system. As the number of people interested in Linux grew, they formed user groups to share information through the Internet with any Linux user in the world. By July 2000, there were more than 400 Linux user groups in 71 countries.

The real fascination with Linux stems from the fact that it is not an organizational project. Instead, volunteers from all over the world contribute code, documentation and technical support because they want to. The first posting regarding the project came on August 25, 1991, when a computer-science student from Helsinki wrote:

"Hi everybody out there using minix – I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386 (486) AT clones. This has been brewing since April, and is starting to get ready. I'd like any feedback on things people like/dislike in minix...."

This was followed by the announcement of Linux v. 0.02 on October 5, 1991. In a message posted to one of the newsgroups on the Internet, Linus Torvalds – the Helsinki student - wrote:

"This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs...and I'm looking forward to any comments you might have."

Furthermore, everybody interested was invited to join the project:

"Are you without a nice project to modify for your needs? Then this post might be just for you...Full kernel source is provided, as no 'minix' code has been used...Sources to the binaries ('bash' and 'gcc') can be found at the same place in /pub/gnu."

Until the year 2000, there were 569 additional releases, all managed and announced by Linus Torvalds who single-handed acts as a filter on all patches and new releases. Depending on his judgement, a contribution can be rejected, accepted or revised. However, to help him in his decisions and in his programming efforts, Linus has an active community of programmers, who, electronically organized, are crucial for advice, suggestions and code. By using Linux mailing lists and Usenet groups, Linux community members get continuously updated on where to send code and where to find information.

In the Linux community, the role structure has been identified as important for the overall organization of the development work. The two most important roles are ‘credited developer’ and ‘credited maintainer’. The ‘credited developer role’ originates from the v. 1.0 release in 1994 in which a credits file was included to publicly acknowledge people who had contributed substantial code to the kernel. The ‘credited maintainer role’ was formally acknowledged in February 1996 when the maintainers file was announced. Designated maintainers are responsible for particular modules of the kernel, for example, they review Linux-kernel mailing list submissions relevant to their modules, build them into larger patches, and submit the larger patches back to the list and to Linus directly.
4.3 Community-based Software Development – the Clusterball Case

Daydream Software is a Swedish game developer. During the time for the study, the company had 65 employees ranging from software developers, graphical designers and web designers to marketing people, administrators and managers. At Daydream, the developers are located in offices close to each other. They communicate face-to-face or by using the telephone, but no major efforts are needed to support their communication and coordination electronically. Instead, the nature of distribution lies in the intention by Daydream to involve its distributed users in the software development process, an intention that was announced by the manager in 1999 in relation to the development of the online game Clusterball:

“Our customers are our best product developers. We want constant feedback on our design suggestions so that we know what they want and how they want the product to improve. We want them as part of the design process.”

While the developers are co-located, the customers are distributed around the world. This posed several challenges to Daydream in terms of communication and coordination tools. To solve this, a virtual community was created in which there was the possibility for users and developers to communicate using electronic forums, chats and e-mail. Very soon, the developers used the community forum to encourage user feedback on one of the beta versions of Clusterball:

“The purpose is primarily to locate configurations that experiences troubles getting Clusterball to run. Please send us feedback on performance and any strange behaviour.”

The response from the community was positive and as soon as the beta-version was made available for downloading, the feedback could be enjoyed:

“I would like to see the ability to set a minimum and maximum player ranking when I host a game. In this way, a ‘Newbie game’ will really be for ‘Newbies’, experts won't come along and thrash everyone.”

“The number one thing I would like to see is demo-recording and playback”.

The postings revealed configuration problems, modifications and future suggestions on additional functionality. Only between July 17, 2000 (official release date) and November 2002 (the end of the study) there were 15,667 postings to the general forum and 1,878 postings to the technical forum. Realizing this, Daydream expanded its organization and appointed a ‘community manager’ to handle the postings and the activity in the forum. Also, the community manager was responsible for communicating the ideas put forward by the community to the rest of the company. According to one of the community members, their influence on the product was significant:

“I think peoples’ suggestions on new features to patches are definitely taken into consideration…the people at Daydream seem to be open to suggestions from us players. “

Also, the developers seemed to enjoy the interaction with community members. In the following statement one of the developers reflects on the benefit of having community members influencing the product:

“I use the community a lot in my work. In reading the postings I always find good suggestions on what to improve. Also, it is fun – I feel like I learn about the customers and what they really want!”
5 Discussion

As illustrated in the empirical cases, there are different approaches to distributed software development that can be taken. Below, the approaches are compared in terms of three dimensions: (1) nature of development approach, (2) communication structure, and (3) coordination mechanisms.

<table>
<thead>
<tr>
<th>Distributed Software Development Comparison Dimensions</th>
<th>Global Software Development (GSD)</th>
<th>Open Source Software Development (OSS)</th>
<th>Community-based Software Development (CSD)</th>
</tr>
</thead>
</table>

### 1. NATURE OF DEVELOPMENT APPROACH

<table>
<thead>
<tr>
<th>Development rationale</th>
<th>Stragetical/Economical</th>
<th>Altruism/Ideology</th>
<th>User participation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development infrastructure</td>
<td>Distributed teams of developers</td>
<td>Distributed individual developers</td>
<td>Distributed individual users</td>
</tr>
<tr>
<td>Role of developer</td>
<td>Producer of software</td>
<td>Producer/consumer of software</td>
<td>Producer of software</td>
</tr>
<tr>
<td>Role of user</td>
<td>Consumer of software</td>
<td>Consumer/producer(^1) of software</td>
<td>Consumer/producer(^2) of software</td>
</tr>
<tr>
<td>Division of work</td>
<td>Hierarchical decomposition of work</td>
<td>Parallel development</td>
<td>User-developer iteration</td>
</tr>
</tbody>
</table>

### 2. COMMUNICATION STRUCTURE

<table>
<thead>
<tr>
<th>Communication type</th>
<th>Developer-developer communication</th>
<th>Developer-developer/Developer-user/User-user communication</th>
<th>Developer-user/User-user communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicative actors</td>
<td>Project managers</td>
<td>Credited developers</td>
<td>Project managers</td>
</tr>
<tr>
<td></td>
<td>Senior executives</td>
<td>Credited maintainers</td>
<td>Community managers</td>
</tr>
<tr>
<td></td>
<td>Senior managers</td>
<td>Community members</td>
<td>Community members</td>
</tr>
<tr>
<td></td>
<td>Subproject managers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Project members</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3. COORDINATION MECHANISMS

| Coordination infrastructure | Face-to-face, phone & audio conference, videoconference, electronic meeting systems, virtual whiteboards, data conferencing | Configuration management systems, web-based bug tracking systems, web-based access to source code, web-based tools for | Virtual community functions such as electronic forums, chats, e-mail |

\(^1\) In terms of software code that can be implemented.

\(^2\) In terms of feedback, design suggestions and modifications to existing design.
voicemail, mail, electronic forums, intranets, e-mail, group calendars

<table>
<thead>
<tr>
<th>Coordination tools</th>
<th>Project management models</th>
<th>Peer supervision</th>
<th>Peer review</th>
<th>Project management models</th>
<th>Development methods</th>
<th>Community postings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Development methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coordination process activities</th>
<th>Requirement specification</th>
<th>Implementation proposals</th>
<th>Software design</th>
<th>Software test and implementation</th>
<th>Software maintenance</th>
<th>Problem discovery</th>
<th>Solution identification</th>
<th>Code development and review</th>
<th>Code commit and documentation</th>
<th>Code release</th>
<th>Software design</th>
<th>Software release</th>
<th>Community-driven test and review</th>
<th>Community-driven modification and maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison of three approaches to distributed software development.

As can be seen, there are significant differences in the nature of development approach. In GSD, there are distributed teams of developers motivated by strategical and economical rationales. In OSS, individual developers contribute without monetary compensation in software projects where the code is freely shared due to altruistic values. Often, projects originate in individual needs, something that is evident in the Linux case where Linus Torvald’s own objective was to create a Unix-like operating system for the IBM PC 386 series (Feller and Fitzgerald, 2002). In CSD, software is produced by developers in close cooperation with distributed individual users in resemblance with the idea of user participation as expressed in for example Participatory Design (Namioka and Shuler, 1993) and Contextual Design (Beyer and Holtzblatt, 1998). These differences are further visible in the ‘roles of developers and users’. While GSD and CSD developers are mainly producers of software, the OSS approach is characterized by developers also being users of the software. Furthermore, OSS, and to some extent CSD, allow for parallel development, something that is not encouraged in GSD.

Due to the diversity in ‘development infrastructure’, there are differences in the communication structure. While the GSD approach focuses on developer-developer communication, both OSS and CSD support communication also between developers and users as well as between users and users. Regarding the ‘communicative actors’, there is evidence of hierarchical structures in all three approaches. This is interesting, especially in relation to the OSS and CSD approaches. While the community culture is often described as a bazaar (Raymond, 1999) where chaotic development processes evolve into coordinated processes, the cases presented in this paper bear evidence of defined structures and hierarchical organizations also within these. In the Linux case, Linus Torvalds dictated the rules, and in the Clusterball case, a community manager was appointed to direct the community activities.
The infrastructure and the tools that are necessary to align distributed developers and distributed users are included in the final dimension of coordination mechanisms. In all approaches, Internet technology is deployed as the infrastructure for coordination. However, while the OSS and CSD approaches depend solely on Internet-based coordination infrastructures, the GSD approach encourages physical interaction and face-to-face meetings. In these, project management models and other formal structures can be negotiated, an activity that is not evident in for example OSS development communities. Rather, these are self-organized and self-governed. Furthermore, there is an interesting difference in the ‘coordination process activities’ that are carried out in each approach. While the design evolves as a result of an iterative process in both OSS and CSD, the GSD approach advocates for a pre-defined process in which requirement specifications direct the process. This is evident in the Kappa case were project specifications were an important part of the project management model.

To sum up, the approaches embrace different characteristics important to distributed software development. While the GSD approach is emerging due to industry and business drivers on the global market (Karolak, 1998), the OSS paradigm can be seen as a provocative, yet fascinating, approach driven by ideological conviction and altruistic interests (Sharma et al, 2002). The third approach, CSD, is adopted by companies trying to infuse features from the community culture into their every-day practices of software development (Sharma et al, 2002). In representing formal and informal organizational structures, and in being approaches that to varying degrees involve the user community in the development process, the approaches offer different opportunities for the range of companies entering the distributed environment of software development.

6 Conclusions

In this paper, three approaches to distributed software development are identified and explored – global software development (GSD), open source software development (OSS) and community-based software development (CSD). Based on a comparison of three empirical cases, it is argued that the approaches embrace different characteristics in terms of: (1) nature of development approach, (2) communication structure, and, (3) coordination mechanisms. In identifying and exploring the three approaches, this research aims at helping both researchers and software developers in:

- Recognizing the different approaches to distributed software development that can be taken
- Recognizing the differences between the approaches, hence, creating an understanding for the types of development situations to which they can be applied

References


