

## Shared mental models among open source software developers

Barbara Scozzi<sup>1</sup>, Kevin Crowston<sup>2</sup>, U. Yeliz Eseryel<sup>2</sup> and Qing Li<sup>2</sup>

<sup>1</sup> Polytechnic of Bari, Department of Environmental Engineering and Sustainable Development

<sup>2</sup> Syracuse University, School of Information Studies

b scozzi@poliba.it, crowston@syr.edu, uyeserye@syr.edu, qli03@syr.edu

### Abstract

*Shared understandings are important for software development as they guide effective individual contributions and coordination of the software development process. However, it is not clear if such understandings can be developed in highly distributed groups that do not regularly meet face-to-face. In this paper, we present the results of a preliminary analysis of shared mental models within a Free/Libre Open Source Software (FLOSS) development team. We analyzed mental models using cognitive mapping and process analysis and compared the models of four developers from the Apache Lucene Java project. Our analysis suggests that there is a high level of sharing among core developers but the sharing is not complete, with some differences related to tenure and role in the project. Finally, we suggest directions for further research on shared mental models in FLOSS teams.*

### 1. Introduction

In response to the problems created by discontinuities in distributed teams, studies stress the need for a significant amount of time spent learning how to communicate, interact and socialize using computer-supported communications tools [1]. In this study, we focus specifically on the role of shared mental models (e.g., common conceptions of the project, other team members, users, competitors or programming standards) that guide team members' behaviours and shape their actions. Research has described the value of shared mental models in conventional teams but as yet we have little evidence of if or how they can be developed in distributed teams that rarely meet face-to-face. The goals of the current study are finding evidence for the existence of shared mental models in one kind of highly distributed teams, namely free/libre open source software (FLOSS) development teams. Specifically, the study addresses these research questions:

1. To what degree are the mental models of core project team members shared? Which aspects are common and which unique to individual developers?
2. What factors (either project or individual) are related to the sharing of mental models? Can we predict which aspects are likely to be shared or not shared?
3. What are implications of shared mental models for team performance?

In this paper, we report on a preliminary analysis of the degree of sharing of mental models among developers in one project. This study is the first step in a study of multiple teams with a larger sample size. This paper demonstrates the general approach, and thus forms a basis for future studies using this approach. The following section presents the theoretical basis for our study. Subsequent sections present the methodology for data elicitation and analysis and our findings. We conclude by discussing advantages and disadvantages of the adopted approach and plans for future research.

### 2. Theory

Shared mental models are defined by Cannon-Bowers & Salas [2] as:

knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and in turn, to coordinate their actions and adapt their behavior to demands of the task and other team members (p. 228).

The issue is not so much whether team members have mental models, but rather the degree of similarity among the models of team members. Shared mental models in teams have been suggested as directly or indirectly related to team effectiveness in theoretical and empirical studies [3]. For example, Walton and Hackman [4] suggested that teams have interpretive functions that allow their members to create a consistent social reality by developing shared mental models. When members have different mental models, group experience can be frustrating, time consuming, and ultimately ineffective [5]. Prior research suggests that the existence of accurate shared mental models that guide member actions are important for team effectiveness [2]. Research on software development in particular has identified the importance of shared understanding in the area of software development. Curtis et al. [6], note that, "a fundamental problem in building large systems is the development of a common understanding of the requirements and design across the project team." They go on to say that, "the transcripts of team meetings reveal the large amounts of time designers spend trying to develop a shared model of the design".

There are several different aspects of mental models that might be shared. First, research suggests that shared interpretive schemes help improve performance in face-to-face teams [7]. Shared interpretive schemes about tasks

and actors' abilities can enable teams to coordinate their activities without the need for explicit communications [8, 9]. The problem of developing shared interpretive schemes is likely to particularly affect FLOSS development, since FLOSS team members are distributed, have diverse backgrounds, and join FLOSS teams in different phases of the software development process [10]. In short, shared interpretive schemes are important as guides to effective individual contributions to, and coordination of the software development process.

A second aspect that might be shared is understandings of expected or allowed behaviours that constitute roles. Prior case studies have described how individuals move from role to role in FLOSS projects as their involvement changes. For example, a common pattern is for active users to be invited to join the core development team in recognition of their contributions and ability. In some teams, this selection is an informal process managed by the project initiator, while others such as the Apache Project, have formal voting processes for new members. On the other hand, we are still learning how the privileges and responsibilities of these different roles are defined. Again, some projects seem to have formal role definitions, while in others, roles seem to be more emergent.

A final aspect of shared mental models that we consider are norms and rules. In the knowledge-based view of the firm, groups are defined as "social entities that hold implicit and explicit rules that guide an individual member's interpretation, contribution and behavior." [11]. Norming is one of the most important steps in group dynamics, so studying norms/rules is crucial to understand the group. [12] notes that rules allow FLOSS developers to form stable expectations of others' actions, thus promoting coordination. The importance of such rules have been documented in conventional software and FLOSS development teams [e.g., 13, 14]. For example, [15] describes a set of implicit and explicit rules for software development in the FreeBSD project (e.g., "Don't break the build"), while [16] notes implicit rules regarding project forking at the community level.

### 3. Data

We next describe how we obtained data for our study, covering in turn our data elicitation approach and subject selection.

#### 3.1. Data elicitation

To elicit data to address our research questions, we interviewed developers active in FLOSS projects. Interviews followed a semi-structured protocol based on the theory above, designed to elicit information on how team members interpret their role and the other members' roles, how they act and the reasons for their behaviours, tacit norms and practices and the way such practices have arisen. Specifically, the interview protocol included:

- *Project rules and norms.* Any explicitly stated norms or rule as perceived by developers.
- *Project environment and constraints.* The environment in which the team operates, constraints that they have to deal with, customers and competitors.
- *Development strategy.* The overall approach to project development.
- *Development process.* Process by which the software is developed (activities, dependencies, coordination mechanisms), tools and technology used for software development, submit and handle bugs, patches and feature requests, and decision-making processes.
- *Team organization.* Team structure and specific team roles.

The decision to adopt a semi-structured protocol was driven by the techniques we decided to use for the analysis, as discussed in the next section.

#### 3.2. Setting and subjects

The results presented in this paper are based on interviews carried out with four developers (of a total of seven committers) affiliated with the Apache Lucene Java project. Lucene Java is a high-performance, full-featured text search engine library written in Java (<http://lucene.apache.org>), suitable for applications that require full-text search. It is a sub-project under the Apache Lucene top-level project.

One author interviewed the four Lucene Java developers attending an Apache-sponsored conference (represented in the rest of the paper by code letters A to D). All four were male. Three of the interviewees were committers in Lucene Java. The committers had different seniority with the project, having been a committer for 2 years (A), 1 year (B) and 3 months respectively (C). The final interviewee (D) was a Project Management Committee (PMC) member for the Lucene top-level project, a contributor to Lucene Java and a committer for another Lucene sub-project (Nutch). The interviews were conducted separately, lasted between 30 and 75 minutes and were recorded and then transcribed for analysis.

### 4. Analysis

Three of the authors separately and then collectively analyzed the interview transcripts adopting an inductive approach. The text of the interviews was carefully read to identify instantiations of the following mental models:

- *Interpretative schemes.* Interpretive schemes include:
  - 1) definitions of key aspects of the projects;
  - 2) causal cognitive maps of project features (e.g., history, key aspects, norms and practices, and organization, in the view of the developers);
  - 3) processes carried out within the projects.
 The analysis process for these elements is described in more detail in subsequent sections.
- *Project roles and resources.* Roles, their responsibilities and privileges and the organization structure.

- *Adopted rules and norms.* Norms are accepted values or ways of behaving. Rules are explicit (hence written) norms.

Some aspects of these structures overlap, e.g., some topics are included in both the process and causal maps, and some project roles are also given in definitions.

#### 4.1. Causal cognitive maps

Causal cognitive maps (hereafter referred to as causal maps) are graphic tools used to represent a person's views of a given set of topics. A causal map is composed of concepts and causal links among them [17]. Concepts represent ideas, opinions and key issues associated to the topic of the map. These are linked by causal relationships, which can be mainly distinguished in cause/effect (which do not imply intentionality) or means/end relationships. Concepts that represent the cause or the means to achieve a given goal are situated at the arrow's tail and concepts that represent the effect or the end at the arrow's head.

Causal maps can be used with different purposes. In this project, they have been adopted for an explicative purpose, i.e., finding evidences of the existence of shared mental models among FLOSS developers working on the same project. In particular, the maps are used to represent and compare the interpretative schemes of the developers so as to gauge the degree of common knowledge as well as to better understand the reasons that underlie team members actions and the dynamics based on which common interpretative schemes, if any, arise.

The causal maps were developed using a technique called Documentary Coding Method [18], which involves the identification of the main concepts cited by the respondents during interviews and the relationships among them. In order to do that, each of the interviews was transcribed and analyzed. Two coders first identified concepts in each transcript. When consistency was not achieved, the authors reviewed the considered text together until they achieved agreement. Afterwards two coders independently drew causal maps based on cause-effect relationships mentioned, and process maps that reflect time-sequenced process relationships. Again, two coders then discussed the differences between their concept maps with each other and with the third coder and based on this discussion, developed a final concept map to depict each interviewee's mental model.

Different methodologies can be used to analyze and compare maps once developed. In most studies, qualitative metrics, e.g., number of heads, tails, domain and centrality are used [19]. Some scholars have also defined ad hoc metrics to compare maps. [20]. Maps can be analyzed and compared by measuring the following qualitative metrics:

- *Heads and Tails.* Map heads are those nodes that only have arrows going in. Heads are representative of the developers' final end/goal and/or the effects of their perception. Tails are those nodes that only have arrows going outside (no arrows go inside). Tails ex-

plain/describe the causes of some perceptions and/or the means adopted to achieve goals.

- *Domain and Centrality.* Domain and centrality provide information about the importance of concepts. In particular, the domain score of a concept is given by the sum of direct links (both as input and output) the attendant node has. The centrality score of a concept is given by the sum of both direct and indirect links the attendant node has, so providing information on those concepts that are often unconsciously considered as the most relevant/central.
- *Sets.* Sets are groups of concepts that deal with a specific issue or topic. Topics were assigned to sets by the authors and again, disagreements about the assignment of topics to sets were resolved by discussion. By counting the number of concepts mentioned for each set it is possible to assess the importance/complexity associated with the topic of the set.

#### 4.2. Process maps

A process is a set of activities that, by using different inputs, carries out an output [21]. Adopting a process view (or a process theory) means explaining how outcomes of interest develop through a sequence of events. A process map is a graphical representation of the process carried out within an organization. Many techniques have been proposed in the literature to map processes [22]. The main objective of such techniques is to gather the information necessary to analyze and, eventually, improve the process. In the paper, process maps were adopted to describe which tasks are accomplished within each project, how and by whom they are performed and which are the dependencies emerging among them. To this aim, by carefully reading the interview texts, we first identified the processes mentioned by developers. Successively, we identified the task, the involved roles and dependencies among task per each process. Finally, we compared the maps of the processes as described by different developers so identifying eventual differences in the tasks, roles and/or sequences mentioned.

#### 4.3. Cross-subject comparisons

Once we had an agreed set of models for each interview, we compared them across the individuals. We first listed and examined items for which a definition was provided and qualitatively assessed the degree of similarity of the definitions. A similar approach was used to compare the view of developers on roles and resources, and norms and rules.

### 5. Findings

In this section, we discuss in broad terms our findings, covering in turn definitions, causal cognitive maps, processes, roles and rules and norms. Table 1 summarizes the number of concepts in each model identified per in-

interviewee and identifies concepts that appeared in multiple interviews (the number of such concepts is reported in bracket). The findings detailed below show that these teams were able to develop shared understandings despite the discontinuities created by having to collaborate virtually, such as the differences in nationalities and cultures of members.

### 5.1. Definitions

A definition is a statement of the meaning of a word or a phrase related to the project or the team. We analyzed definitions and instantiations of those words and concepts separately. The analysis of the definitions provided by the interviewees are provided in Table 2. An important point is the high degree of sharing of key definitions, such as project goals, users and challenges. For example, both senior members (A and B) mentioned that the team does not have clearly stated goals, but yet the community works towards the same goals. As one of them put it, "It's really kind of a free flowing communal meeting of the minds". This is also evident in that all three members identified the goal as developing a search library. Only the non-committer member described the project as "information retrieval" project. Similarly, three interviewees described the intended users as people who want to incorporate search into their applications. When asked for challenges related to the project, all members were able to identify some challenges, yet, it seems that none of them take those as "problems". All three members clearly stated that "there aren't any big problems in the project".

An interesting difference is in the descriptions was in the area of the challenges that the new members face. The most senior committer A thought new members did not face any particular challenges, since they will have been part of the community for a while before becoming a committers. On the other hand, the second most senior committer, B, suggested the challenges of getting up to speed on Apache infrastructure, commit rights, and so on. The newest committer, C, identified four different challenges, such as the burden of suddenly being responsible and writing a good code, as well as trying to work with non-committers and encouraging them to submit patches. Finally, D thought the project is complex to jump in, so people need to go through quite bit learning. Meanwhile, he also has an opposite opinion: the barrier to enter is low.

### 5.2. Causal cognitive maps

Analysis of causal cognitive maps requires the comparison of causally connected concepts. The causal maps for the interviews included a large number of concepts (ranging from 63, in the case of D, to 153 in the case of B) but with only a few causal connections. As the maps were not very well connected, the heads and tails analysis did not yield interpretable results.

We turn next to a comparison of the causal maps examining central concepts. Many concepts with the highest

domain scores also have the highest centrality scores (Table 3 shows concepts with the highest domain score), suggesting which concepts are considered as the most relevant by each developer. Concepts that have the highest centrality scores (but not the highest domain scores) are associated to: the abilities of new members and new committers and the reasons to take part to the project for A; some project strengths and some aspects of the Apache project for B; some project strengths in the case of C. Finally, for D they are the fact Lucene Java is used in many projects, the project founder's contribution and the committers' mindset.

The success of Lucene Java is central (both in terms of centrality and domain), thus most relevant, in three maps. However, apart from it, relevant concepts differ from developer to developer. For example, A describes the abilities of another committer, and also mentions a step of the procedure to become a committer. B talks of a step of the new member hiring procedure, C mentions some project strengths and a step of the committing procedure, Finally, D talks of the different modes to contribute to the project, the extension of the community and new members' main issue and problems.

Finally, by examining the concepts presented in the maps, nine sets were identified. Whereas the definitions reflect the interviewees' point of view, the sets are the classification of the concepts based on the interpretations of the researchers. The sets are not mutually exclusive. The sets are:

1. Challenges of the project: Challenges faced by the members in working on the project.
2. Challenges of new members: Challenges that are faced by developers when they become a committer.
3. Community: Statements on how community gets along, etc.
4. Roles: Number of members, and the role of the community, which may include formally identified roles, informal (perceived) roles and specific definitions of individuals roles.
5. Coordination: How coordination issues are addressed.
6. Goals: Goals of the project.
7. History: How the project was initiated and the growth stages that the project went through
8. Leadership: Who the leaders are and why they are identified as leaders.
9. Membership: New member selection
10. Skills and Knowledge: Skills and the knowledge needed by the members to contribute to the project as a committer.
11. Success Factors: Why the project is identified as successful (or not).
12. Strengths: Strengths of the software and the team.
13. Communication: How members communicate.
14. Rules/Norms: The rules that are formally stated or written using project website or mailing lists as well as unstated rules that are widely accepted (norms)
15. Change in project: Change in the project over time.

16. Personal motivation: Reasons why interviewees chose to join and work actively on the project.

The relevance of each set (i.e., the number of concepts per set) is an indication of the importance that different issues have. As shown in Table 4, issues related to community are the most cited by three developers. Challenges and success/strengths are the most important issues for the two of them.

### 5.3. Processes

We turn next to a consideration of the processes identified by the interviewees. Three of the interviewees described two-three processes whereas one described in detail seven different processes (Table 5). However, this difference might be attributed in part to a longer interview for this informant.

All members had the same understanding of how the project was initiated in terms of the main process steps and their order: project initiation by the owner in SourceForge and then the move to Apache. Yet, two of the interviewees provided more details at the end of this process. Interviewee A suggested that the project first became a top level project and then Lucene Java became a subproject when Lucene Nutch came on stage. Interviewee D also mentioned that before the project became top-level, it was a subproject under Apache Jakarta.

Although all of the interviewees described the member selection process, they described it at different levels of detail. All interviewees mentioned that one person nominates a candidate and then PMC votes on the membership. Three interviewees suggested multiple criteria for someone to be nominated that include high quality contribution over a long period of time and making positive comments on the mailing list. Interviewee C mentioned criteria of selecting members as “having specialty in certain area, being active for a while, submitting good patches and being responsible enough.” Also he identified the required skills as “knowledge in search, java and knowing how open source works”. B mentioned the “litmus test” as “contributing high quality stuff, for a time period”, as well as “cordial to each other”. He listed skills as “java, personality skills, specialization, knowing where to look in other parts of code, knowing how to ask, knowing what you know and don’t know”. A mentioned that to “become expert level users, [one should] contribute patches, [and] not be abrasive.” He also mentioned the PMC as “a step over committer” who “made a lot of contribution and sustained”.

The release process and patch processes are also described by two of the interviewees. We will first discuss the release process. Both members used two steps in the process and they focused on totally different aspects of releases. One interviewee focused on how the release decision is initiated, by mentioning that one person suggests that they should do a release and then the team discusses when it should be done and what features should be included. Another interviewee focused on how the members

contribute to the decision of doing the release or not. He identified the voting process and the ability to hold off the release if a developer suggests that something is missing or they are working on a section that should be included in that release.

The patch process was also mentioned by two members. While interviewee B provided a detailed process with 10 steps, including 2 decision points, C provided a 3-step process that assumes that patches are submitted by non-committers and includes a single decision point. C described that non-committers would submit a patch for the features they need, a committer who feels comfortable in the area of patch would hold on to it, and commit it if the patch is very thorough. Interviewee B mentioned the process from the idea generation to commitment of the patch. He suggested that first, the ideas are received on the listserv. Easy ideas are then simply implemented by someone whereas more complicated ideas involve discussion and identification of backward compatible suggestions. Then somebody submits an enhancement report, a developer then submits a patch and if there is a test for the patch and there’s no discussion on the patch, one of the developers will just commit it. Otherwise there might be a back-and forth discussion on the patch and perhaps a cooling-off period before a final decision.

### 5.4. Roles

We turn now to a consideration of the roles identified. Various roles were mentioned by the interviewees: 1) formal roles, which are also documented on the project website, and 2) informal roles, such as users, and the members of the community. Interviewees also provided examples of the mentioned roles. In this section we will first talk about the formal roles, and then talk about the informal roles with emphasis on the role of the community and the role of specialization in the project. Lastly, we will discuss leadership roles in the project.

The Lucene website shows two formal roles: Project Management Committee (PMC) and committers. The website presents committers at three levels: core committers, contributing committers and emeritus committers.

All interviewees recognized that some members in the community are in the core and some are peripheral, and that different people work on different aspects of the project, such as coding, or answering questions on email-lists. All of them emphasize the importance of community and consider it as one of project strengths and success factors. Compared to D and C, A and B provided a more overall view of community. A said “it is a viable sustainable community”, because of the increasing interest and usage from both companies and individuals. A also considers new committers as the core developers doing a large amount of coding at different time. B said “it is vibrant community”, which contributes to project success. Same as A, B emphasized the importance of new people and clearly connected the spike of group activity with new committers.

When defining the role of the community, B, C and D offered similar opinions. B referred to the role of the community as “they provide feedback about what works and what doesn’t work”. When referring to the community role C said “users give feedback how they use Lucene. We answer the questions. They ask for new features”. D said “one big strength is the community. There are lot of people who know that Lucene very well and are kind of supporting everybody who is trying to use it”.

Interviewees also had interesting comments that help us understand the leadership dynamics in Open Source. Two senior interviewees mentioned that “the project founder will always sort of be the head [leader] still”. The project founder is not as active in Lucene Java currently. Yet, “his opinion carries a lot of weight in the community” as stated by two interviewees. They also mentioned that he limits his comments not to influence the community. On the other hand, two interviewees identified other leadership dynamics in the team. The most senior committer suggested that there are either no leaders at this time or multiple leaders. He also suggested that there might be leadership in certain parts of the project, as well as a leader from the overall organizational perspective. He suggested that he might be one of the leaders due to the work he did the previous year as well as the organizational work (such as getting releases) that he did. In fact, the second senior committer also identified him as one of the two current leaders. According to these two interviewees, leadership seems to be correlated with sustained contribution. On the other hand, the third committer perceives leadership negatively, as almost being close to dictatorship, and thus identifies no leader. He says “there are no leaders in the team, everybody has a say and rights”.

### 5.5. Norms/rules

Finally, we consider some norms expressed in the interviewees. Of the four interviewees, only B clearly described many norms and rules, while the others only briefly mention a few unspoken norms when describing certain processes. The norms mentioned can be put into five categories: community relationship, communication, decision-making, membership and coordination.

In the norm of “community relationship”, two members identify that members should be cordial to each other in interactions. A lists “nice enough” as an important criteria to select new members. Also, they try to avoid friction by really convincing people, because they think the community is very important. B points out the rule directly about “we all try to be cordial with each other.” And he attributes it to the project founder since “he brings a lot in this area”. Also, both A and B mention a factor leading to the group cohesion, because people defer to others in certain areas. Everyone has some deep understanding and specialize in different component. So even all members have permission to commit any codes they want to commit, they will prefer to let others do due to their specialties.

About “communication” norms, three members point out most of conversations use public email list when “it comes to Lucene itself”. This norm is consistent with the Apache rule that tries to involve the whole community in the group work and discourage offline behaviors.

The norm about decision-making is mentioned consistently by all four members. They point out “everybody has a say and right”, so there is no person who always has the last word. However, two members recognize the weight carried by Doug. A mentions “he will let things play out by themselves a lot... since he doesn’t want to influence things” and B also points out “if he said ‘no I don’t think we should do that’, then that would probably be the final word.” They mention the weight gain by actual contribution as well. Both A and D mention if somebody “does a significant amount of work” in certain areas, he will have more says in those parts. Though everybody has a say, two members mention there is binding vote which can be only done by PMC member. According to B, “if anybody votes minus one, that is binding across all the committers.” While there are no explicit criteria about how to recruit new members in the group, they do express some shared understandings. It is clearly expressed by B as “The Litmus test”, which is “usually have you been contributing high quality stuff, and have you been doing a time period. And whatever is that time period is fluid but it’s probably been more than a month. But it might not be that long. It’s definitely more than a week.”

Coordination norms constitute the major part of normative system in a traditional group. But in open source context, few of them are identified. Most of members mention there is no much planning in the group. Sometimes, they plan only for “big issue”, rather than routine work. Only B identifies a lot of interesting coordination norms to guide the member’s behavior in an implicit way in every aspect of the project. For example, on the topic “patch submission”, B states a norm “we pretty much won’t accept patches that don’t yet have a test for them”. After putting in a patch in the list, “if there is really no discussion on that [5 days period], we agree to it [to be committed].” Besides process type norms, B also mentions a task type norm about the quality of code, “you should benchmark your changes to see whether you made things slower or faster. And if it’s slower you probably shouldn’t be committing it unless it was really a bug.”

From these interviews, we can see there are very few explicit rules in the team, which is mainly guided by unspoken norms. Due to their implicitness, it is not easy to observe norms embedded in people’s mind. Even the interviewees themselves may not realize what norms they follow. However, some fundamental norms crucial to the group effectiveness are still recognized by multiple individuals, such as community relationship, communication and decision making. Moreover, even if they express the norms at various level, it is very interesting to see that those norms stem from some underlying espoused values and beliefs which are clearly shared by group members. For example, one of the most important values is “Com-

munity is important.” A fair number of norms follow from this belief, such as avoiding conflicts, respecting peers and acknowledging others’ contribution.

## 6. Discussion

Our findings, as reported above, do show a degree of sharing of mental models, which is different than the notion that virtual teams may lack shared understandings due to discontinuities and national and cultural differences found in virtual teams. As to the interpretive schemes, key definitions (e.g., project goals, users and challenges) have a high degree of sharing among the core developers we studied. Some aspects of the cause maps are shared as well. For example, concepts related to project success/strengths, challenges and the role of community are central and/or relevant in most maps. As to roles, the importance of community is stressed by all members. Finally, norms and rules show a high degree of sharing.

However, some differences in the views of developers also emerged. For example, Interviewee D, who is a PMC member in the Lucene Project but who is more actively involved in Lucene Nutch instead of Lucene Java had less shared understanding than those who are not PMC members, yet involved directly with Lucene Java.

The shared mental models about the rules and norms as well as the focus on the “community” aspect seem to closely reflect the overall “Apache” way, indicating that models may be shared at different levels: among members of an individual project, the Apache Foundation or perhaps even the FLOSS movement as a whole. In looking for the source of these models then, we need to consider both individual interactions and the values promoted by organizations, even for these self-organizing and voluntary project teams.

### 6.1. Benefit and drawbacks of causal maps

Since one goal of this paper was to test methods for use in a broader study, we offer some reflections on the use of causal maps. The main benefit that derives from the adoption of the maps is the ease of the analysis of different perspectives. The graphical representation facilitates identification of the key issues and the differences among different positions. Moreover, the adopted metrics facilitate the understanding of concepts or relationships not perfectly clear or conscious to individuals. These relationships can be more easily stressed than is the case when other qualitative tools (such as case studies or simple interviews) are used. The focus on causal relations sets causal mapping apart from techniques such as a repertory grid, which provides a description of constructs and personal values from interviewees’ perspective, but with less attention to the relationships between constructs.

Of course, causal maps also present some drawbacks. In particular, the stage of the knowledge elicitation (interviews and codification of collected data) is the most critical. As most of the qualitative research methodologies,

the knowledge schemes of the interviewer (i.e., the researcher) can strongly influence the findings. By knowledge scheme we mean the culture, interests and experiences of the interviewer. The researcher’s knowledge scheme can influence the way questions are asked (so influencing the answers) and, above all, the way data are analyzed. This issue poses a particular problem for creating cause maps that can be meaningfully compared across subjects. It was difficult to conduct semi-structured interviews that let respondents choose topics they felt were important while ensuring that they talked about the same items.

There exist some techniques that try to reduce the subjectivity, but they introduce other sources of error [20]. For example, by providing an ex-ante defined list of possible constructs and concepts (though in some cases they can be extended by respondents) the answer possibility of the respondents is limited and can be biased. Based on our previous experience, we have decided to adopt semi-structured interviews so trying to minimize the effects of biases. Despite the drawbacks, we argue that causal maps can be effectively used to characterize the interpretative schemes of the FLOSS team members as well to assess if such schemes are shared and how they affect work practices.

## 7. Conclusions

We have presented a preliminary analysis of the degree of sharing of mental models among developers in a FLOSS development project. Our analysis suggests that there is a high-level of sharing among core developers but the sharing is not complete, with some differences related to tenure in the project. Our future work will extend these results in several directions:

1. Attempt to validate the models by seeking feedback from the developers involved.
2. Include more developers from the Lucene Java project, including those with different degrees of participation in the project in order to assess the degree of sharing and how the sharing relates to individual characteristics. In particular, the four subjects analyzed in the current paper all attended the ApacheCon conference, indicating a strong interest in the project. However, attendance may also be a mechanism for developing models, so we need to compare their models to those of developers who do not attend.
3. Include developers from other projects. We have completed interviews with developers from several other Apache projects that we will analyze to assess the degree of sharing between different projects. We anticipate finding certain concepts in common but others that are unique to the particular project. We would also like to add non-Apache projects to see the influence of the Apache Software Foundation on sharing of mental models.
4. Include developers from less successful projects. The projects selected so far have all been rather success-

ful. We would like to interview developers of a failing project to determine if there is a relation between the degree of shared mental models and project effectiveness.

## References

- [1] B. Butler, L. Sproull, S. Kiesler, and R. Kraut, "Community effort in online groups: Who does the work and why?," in *Leadership at a Distance*, S. Weisband and L. Atwater, Eds. Mahwah, NJ: Lawrence Erlbaum, 2002.
- [2] J. A. Cannon-Bowers and E. Salas, "Shared mental models in expert decision making," in *Individual and Group Decision Making*, N. J. Castellan, Ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1993, pp. 221-246.
- [3] J. R. Rentsch and R. J. Klimonski, "Why do 'great minds' think alike? Antecedents of team member schema agreement," *Journal of Organizational Behavior*, vol. 22, pp. 107-120, 2001.
- [4] R. E. Walton and J. R. Hackman, "Groups under contrasting management strategies," in *Designing Effective Work Groups*, P. S. Goodman and Associates, Eds. San Francisco, CA: Jossey-Bass, 1986, pp. 168-201.
- [5] L. L. Levesque, J. M. Wilson, and D. R. Wholey, "Cognitive divergence and shared mental models in software development project teams," *Journal of Organizational Behavior*, vol. 22, pp. 135-144, 2001.
- [6] B. Curtis, D. Walz, and J. J. Elam, "Studying the process of software design teams," in *Proceedings of the 5th International Software Process Workshop On Experience With Software Process Models*. Kennebunkport, Maine, United States, 1990, pp. 52-53.
- [7] J. Sutanto, A. Kankanhalli, and B. C. Y. Tan, "Task coordination in global virtual teams," presented at Twenty-Fifth International Conference on Information Systems, Washington, DC, 2004.
- [8] J. A. Espinosa, F. J. Lerch, and R. E. Kraut, "Explicit versus implicit coordination mechanisms and task dependencies: One size does not fit all," in *Team cognition: Understanding the factors that drive process and performance*, E. Salas and S. M. Fiore, Eds. Washington, DC: APA, 2004, pp. 107-129.
- [9] K. Crowston and E. Kammerer, "Coordination and collective mind in software requirements development," *IBM Systems Journal*, vol. 37, pp. 227-245, 1998.
- [10] L. Gasser and G. Ripoché, "Distributed Collective Practices and F/OSS Problem Management: Perspective and Methods," presented at Conference on Cooperation, Innovation & Technologie (CITE2003), University de Technologie de Troyes, France,, 2003.
- [11] H. Annabi, "Moving from Individual Contribution to Group Learning: The Early Years Of The Apache Web Server," in *School of Information Studies*. Syracuse, NY: Syracuse University, 2005.
- [12] M. A. Rossi, "Decoding the 'Free/Open Source (F/OSS) Software Puzzle': A survey of theoretical and empirical contributions," Università degli Studi di Siena, Dipartimento Di Economia Politica, Working paper 424, 2004.
- [13] S. Sawyer, "A Social Analysis of Software Development Teams: Three Models and their Differences," presented at The 2000 Americas Conference on Information Systems (AMCIS 2000), 2000.
- [14] K. J. Stewart and S. Gosain, "Impacts of ideology, trust, and communication on effectiveness in open source software development teams," presented at Twenty-Second International Conference on Information Systems, New Orleans, LA, 2001.
- [15] N. Jørgensen, "Putting it all in the trunk: incremental software development in the FreeBSD open source project," *Information Systems Journal*, vol. 11, pp. 321-336, 2001.
- [16] E. S. Raymond, "Homesteading the noosphere," *First Monday*, vol. 3, 1998.
- [17] M. Pidd, *Tools for thinking modeling management science*. Chichester: John Wiley and Sons, 1996.
- [18] M. T. Wrightson, "The documentary coding method," in *Structure of Decision*, R. Axelrod, Ed. Princeton: Princeton, NJ, 1976, pp. 291-332.
- [19] P. Cossette and M. Audet, "Mapping of an idiosyncratic schema," *Journal of Management Studies*, vol. 29, pp. 325-347, 1992.
- [20] L. Markoczy and J. Goldberg, "A method of eliciting and comparing causal maps," *Journal of Management*, vol. 21, pp. 305-333, 1995.
- [21] H. J. Harrington, *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness*. New York: McGraw-Hill, 1991.
- [22] V. Grover and W. J. Kettinger, "Business Process Change: Concepts, Methodologies and Technologies." Harrisburg: Idea Group, 1995.

**Table 1. Concepts and shared concepts in mental models.**

<b>Elements</b>	<b># of identified elements</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
Definition concepts		40	34	37	10
Definition concepts repeated		It is written in Java (3); [Goal is to] provide search functionality (3); [Users are] people who want to add search functionality into their applications (3)			
Causal cognitive map concepts		119	153	72	63
Processes		3	7	3	2
Processes repeated		Process Initiation (4); member selection (4); bug-fixing-feature adding (2); release kick off (2).			
Unwritten rule (norms) concepts		5	14	5	4
Unwritten rule concepts repeated		Community is important (3); Communication is done on public mailing list (3); [New member selection criteria] (3) No formal goals, yet same direction (2); not much planning (2)			
Written rule concepts		2	4	3	3
Written rule concepts repeated		Voting for new member selection (4); Accepting patches (3); PMC Role (2); no formal roles (2)			

**Table 2. Concepts per definitions identified**

<b>Definition Elements</b>	<b>Identified # of elements</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
Project description/characteristics		7	4	6	1
Project description/characteristics repeated		Written in Java (3); search library (2); easy to use (2)			
Project goals		3	2	2	2
Project goals repeated		Provide search functionality (3); there aren't specifically stated goals, yet we're driving in the same direction (2)			
Intended users		2	1	1	1
Intended users repeated		People who want to add search functionality into applications (3); developers (2)			
Project success and strengths		5	10	7	4
Project success and strengths repeated		Number of users (2), companies using it (2); community (2); developers who support it (2); interest at the Apache Conference (2); software performance (2)			
Problems faced by the project		2	2	2	1
Problems faced by the project repeated		There aren't big problems (3)			
Challenges faced by new members		1	1	4	0
Challenges faced by new members repeated		None			
Skills/knowledge needed for development of Lucene Java		2	6	3	1
Skills/knowledge repeated		Information search/retrieval (2); java (2)			
Roles		8	6	10	N/A
Roles repeated		Specialists (2); active committers (2); new committers (2); people who support users by answering emails (2)			
Leadership		10	2	2	N/A
Leadership repeated		The project founder will always be the leader (2); Yannik is a leader (2); leadership equals sustained contribution (2)			

**Table 3. Central concepts per map.**

<b>Subject</b>	<b>Concepts (domain scores)</b>	<b>Summary of Concept Areas</b>
A	Erik is good for a lot of projects (4) [another project leader] is important to get others to use Lucene (4) [New members from IBM] can handle lot of these things (4) They know how to fit in (3) Somebody ends up nominating them (3); I have slacked up (3) You want to work on a project (3)	Boundary Spanning Selection of & contribution by new members. Motivation for project
B	One of the most successful open source projects there is (19); there's one of two ways to select new members (5)	Project success New member selection
C	And then they commit it (3); I don't see any problems (2); It is very successful currently (2); Main focus should always be on simplicity (2); That's why I learnt Open Source (2); You can trust (2); So that the committers have a good feeling that the code is good and it's robust (2); Sometimes people of the other [sub]projects get involved in discussions and ask us to implement new features in a way to keep files and docs compatible (2); It's very simple to get basic search working but it also offers more sophisticated stuff for who are familiar with info retrieval and search (2)	Project success, strengths and challenges Process for committing code Motivation for project
D	I would say [the project is a successful one] (2); There is a great number of people involved in the community (2); It either requires you to go through lots of learning to get to a level where you are actually not able to improve it (2); So, it is kind of complex to start (2); The barrier to enter is kind of low (2); It is not a project that you can just jump in and do all the hard stuff (2); Contribution to positive group atmosphere, resolving conflicts, things like that (2); There are so many ways you can contribute (2); So even if you're very entry-level Lucene guy, you can contribute by helping others (2)	Project success & strengths Community & contribution Challenges for new members Group Maintenance

**Table 4. Number of concepts for each set per map.**

<b># of concepts</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<b>Sets</b>				
Challenges	1	6	10	4
Change in Project	6	2	4	3
Community	55	23	7	7
Coordination	4	5	9	2
Goals	5	2	4	1
History	7	3	2	3
Leadership	16	2	3	3
Membership	20	19	9	3
Success/Strengths	2	29	17	3

**Table 5. Activities per processes described.**

<b># of activities</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<b>Processes</b>				
Initiation	4	4	3	4
Rulemaking	-	8	-	-
Bug fixing- Feature adding	-	10	3	-
Member Selection	6	5	4	2
Planning	-	4	-	-
Release kick off	2	3	-	-
Apache Incubation Process	-	4	-	-