11-25-2011

Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development

B Veeresh Thummadi
Case western reserve university, vxt42@case.edu

Kalle Lyytinen
Case Western Reserve University, kalle@case.edu

Sean Hansen
Rochester Institute of Technology, shansen@saunders.rit.edu

芽|Sprouts Working Papers on Information Systems ISSN 1535-6078

Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development

B Veeresh Thummadi
Case western reserve university, USA

Kalle Lyytinen
Case Western Reserve University, USA

Sean Hansen
Rochester Institute of Technology, USA

Abstract
Requirements have been the culprits for budget overruns and failures in software development projects. Fixing the requirements in the early stages of a project can dramatically reduce recurring costs. Past research has focused on linear sequential requirements activities as a means to fix the requirement problems. This line of thinking has led researchers to overlook the possible solutions to requirement problems in social, cognitive, and organizational factors. We probe the success of open source software development and its implications for the linear approach to requirements activity. Despite a wide scale distribution of requirements knowledge among people and artifacts, open source projects have been able to manage and evolve requirements in an organic way leading to high quality outcomes. Even though such efforts include little emphasis on explicit quality in RE practices, these projects often come up with software that meets high quality requirements. In order to understand this anomaly in open source software development, we apply the theory of distributed cognition to understand how social, structural, and temporal dimension impacts the quality of the requirements.

Keywords: open source software development, distributed cognition, traditional software development

Permanent URL: http://sprouts.aisnet.org/11-151

Reference: Thummadi, B.V., Lyytinen, K., Hansen, S. (2011). "Quality in Requirements Engineering (RE) Explained Using Distributed Cognition: A Case of Open Source Development ," Proceedings > Proceedings of JAIS Theory Development Workshop . *Sprouts: Working Papers on Information Systems*, 11(151). http://sprouts.aisnet.org/11-151

Quality in requirements engineering (RE) explained using distributed cognition: A case of open source development

Abstract

Requirements have been the culprits for budget overruns and failures in software development projects. Fixing the requirements in the early stages of a project can dramatically reduce recurring costs. Past research has focused on linear sequential requirements activities as a means to fix the requirement problems. This line of thinking has led researchers to overlook the possible solutions to requirement problems in social, cognitive, and organizational factors. We probe the success of open source software development and its implications for the linear approach to requirements activity. Despite a wide scale distribution of requirements knowledge among people and artifacts, open source projects have been able to manage and evolve requirements in an organic way leading to high quality outcomes. Even though such efforts include little emphasis on explicit quality in RE practices, these projects often come up with software that meets high quality requirements. In order to understand this anomaly in open source software development, we apply the theory of distributed cognition to understand how social, structural, and temporal dimension impacts the quality of the requirements.

Keywords: open source software development, distributed cognition, traditional software development

1. INTRODUCTION

The identification and management of software requirements have been a persistent challenge in the field of information systems development. Despite significant growth in the field of requirements engineering (RE), requirements still remain problematic (Roman, 2006). Most requirement issues arise due to lack of clarity or completeness(Sommerville & Ransom, 2005). Lutz (2002) showed that more than 60% of software errors are caused by errors in requirements. In addition, Boehm suggests that fixing errors in the software implementation stage can be 100 times more expensive than fixing the errors in the early stage activities of requirements determination (B. Boehm, 1983). Especially in large scale systems, ambiguity and inconsistency in requirements can often make the resulting system useless.

At the same time we witness that open source development - "a production model that exploits the distributed intelligence of participants in Internet communities"(Kogut & Metiu, 2001) - has been able to come up with large software systems that are functional and of high quality (Weber, 2004) to the extent that many open source products have challenged the market position of the commercial software platforms (Moody, 2001). In addition, most of the open source projects operate under wide scale geographic and temporal distribution. This begs the question: how is it possible that the open source projects can maintain a high quality in their requirements despite significant heterogeneity in terms of knowledge, people, skills, artifacts, and locations?

We posit here that the traditional idea of requirements captured and documented early in a single document limits our understanding of the critical cognitive aspects of the requirements process that can make them successful. Such cognitive processes are important in that they offer a holistic perspective of the underlying processes that affect system attributes and outcomes. In this paper we approach RE process as a holistic cognitive system based on theories of distributed cognition (Nardi, 1996). Hutchins (1989) proposed an approach called 'distributed cognition' to understand cognition in social context for rendering it an ecological interpretation (Hutchins & Lintern, 1996). We will use this theory as it fits well in understanding how the cognitive processes entwine in designing successfully complex open source systems under wide scale distribution. This theory specifically focuses on (1) how the knowledge gets transmitted and maintained among team members when engaging in a complex cognitive task; and (2) how this knowledge gets distributed and propagated among artifacts at hand among the team members (Rogers & Ellis, 1994)

2. LITERATURE REVIEW

2.1.Requirements Engineering (RE)

Requirements have always been difficult to characterize succinctly(B. Cheng & J. Atlee, 2007). Some researchers feel that requirements define what a system is supposed to do rather than how it should be done. Sommerville & Sawyer offer this notion of requirements when they state that "requirements limit the designer's future behaviors by clearly articulating the requirements of what the system should not do"(Sommerville & Sawyer, 1997). Hence they define 'requirements' not just merely as problem statements but as a mixture of problems, system behaviors, and other design considerations(Sommerville & Sawyer, 1997). Pohl augments this argument by defining the requirements process as an "iterative co-operative process" which analyzes problems and document changes over multiple representation formats for improving the understanding of the requirements(Pohl, 1993). Jirotka and Goguen provide a different perspective by viewing requirements as properties of the system which needs to be possessed in order for the system to succeed in a given environment (Jirotka & Goguen, 1994). They reinforce the importance of both social and technical aspects in requirements by explicating that the software should succeed in a set of given social and technical environments. Bergman et al.(2002) offer an additional characteristic for requirements in large scale software projects as being "inherently political"(Bergman, King, & Lyytinen, 2002). All the characteristics of requirements described above invite a more holistic approach for tackling the real world problems during software development.

System designers have therefore applied system engineering techniques early on to ensure that the requirements are complete (i.e. they cover all environments and critical elements therein), consistent (i.e. they do not pose contradictory demands or assumptions about the environments) and relevant (i.e. they are critical for the survival of the software). Lutz & Boehm reinforce the importance of fixing the requirements early on by drawing attention to the skyrocketing fixing costs in implementation stages (B. Boehm, 1983; Lutz, 2002). Therefore, activities that contribute to higher quality requirements are economically important as they affect both the cost of delivering and cost of (not) using the software. To indicate the importance of managing requirements related knowledge in software development a term 'requirements engineering' (RE) has been coined to encompass all the activities during software development which involve ‘computing’ the requirements of a system (Sommerville & Sawyer, 1997). Requirements engineering(RE) can be more formally defined as a process by which the requirements are gathered, formulated and monitored(B. H. C. Cheng & J. M. Atlee, 2007).

2.2.Requirement quality

Despite a significant growth in techniques of requirements engineering, requirements still suffer issues with consistency, completeness, feasibility and testability(B. W. Boehm, 1984; Roman, 2006). In software development projects, the clients are not sure what they exactly want before using the system which creates the conundrum of “catch-22” and hence fall into the trap of generating inconsistent requirements. At the same time, clients’ uncertainty in estimating requirements can produce incomplete and infeasible requirements(Bell & Thayer, 1976; B. W. Boehm, 1984). In addition, insufficient understanding of the stated requirements by the developers can produce software which is useless.

For addressing the pitfalls of requirements quality, researchers have developed several techniques which are either qualitative (Mylopoulos, Chung, & Nixon, 1992; Robinson & Fickas, 1994) or quantitative in nature (Keller, Kahn, & Panara, 1990). Qualitative approaches use negotiation techniques like house of quality principles or predicate logic (Chung & do Prado Leite, 2009; Liu, 1998; Mylopoulos, et al., 1992; Robinson & Fickas, 1994) while quantitative approaches rely on metrics for evaluating the quality of the requirements(Liu, 1998). The main limitations of these approaches are that they are either subjective or objective. Hence relying on just one approach may not help in addressing the core issues of the requirements quality.

Most of the quantitative approaches described above have used software requirement specification (SRS) documents for measuring the quality of the requirements. Further, researchers have emphasized on 24 distinctive dimensions of quality for measuring quality in SRS (Davis, et al., 1993). Yet, some researchers have argued that quality attributes that really matter are consistency, completeness feasibility, testability as they encompass most other requirement attributes(S. W. Hansen). Hence, we chose consistency, completeness, feasibility and testability attributes in understanding the quality of the requirements in open source. The table 1 below shows the definitions of the quality attributes that we will be using for this study.

Table 1: Quality attributes and their definitions for requirements

Quality attributes	Definition
Completeness	Requirements specification is complete if all the parts are "present" and "fully developed".(B. W. Boehm, 1984; Roman, 2006)
Consistency	Only one possible interpretation of the requirement specifications(B. W. Boehm, 1984; Roman, 2006).
Feasible	Functional and non functional requirements can be met in real time without exceeding the proposed costs by identifying the high risk issues(B. W. Boehm, 1984).
Testability	Requirements stated can be examined precisely to check if the developed software meets the prescribed specification(B. W. Boehm, 1984).

2.3.Requirements management in waterfall,agile and open source development

The RE process can be quite different across different types of software projects in terms of how the requirements are managed to ensure that requirements have adequate quality. Differences can be caused by 1) the timing when the requirement emerges, 2) the way in which it is expressed and where it is coded and stored; 3) who controls it and how it is chosen; 4) and how this knowledge is used to guide downstream development.

In the case of traditional software projects, the requirements are assumed to emerge early, they are coded in formal requirement specification (with strict standards stating the representation) called software requirements specification (SRS) document, which then acts as baseline to manage requirements quality; it can be stored in a formal requirements management system like Doors, it is controlled by a project manager and the client who chose it through a contract, and the process makes sure through requirements tracing that the requirement is met in the final system (Sommerville & Sawyer, 1997). Accordingly, the management process evolves through specific phases like elicitation, analysis, specification, validation and management to ensure that the requirements are complete, consistent and relevant (Sommerville & Sawyer, 1997).

Likewise, open source[1] communities, have more informal ways to manage requirements (Scacchi, 2002). One of the main differences that exist between open source and traditional software development is also the way in which requirements are stored and expressed. In open source development requirements are mainly expressed, shared and stored through constant interactions that are recorded in electronic bulletin boards, e-mail threads and chats until a final implementation is released (Scacchi, 2002). In addition most of the knowledge flows related to requirements are computer mediated and virtual; while traditional software projects relies heavily on face to face communications for gathering and expressing requirements early on until they are recorded in a document. As these RE management process are different we tabulated them to show the differences

[1] Open source term has been used in different ways to refer free and open source software (OSS), free/open source software (F/OSS), and free/libre and open source software (FLOSS) which are different in the licensing terms. When we refer to open source, we adopt the definition of Von Hippel & von Krogh to open source as free software for which the source code is available(Von Hippel & Von Krogh, 2003).

that exist in traditional and open source along four dimensions: timing, communication & storage, control, use of RE knowledge (see Table 2 for more details).

Table 2: Comparing RE practices in software methodologies

	Traditional	Open source
1)Timing of requirement	Client/business needs, changes in the market	Developer/user needs, industry standards
2)Communication & storage	Face-to-face, SRS documents	bboards, forums, email-threads
3)Control	Client, Project manager, developers	Core developers
4)Use of RE knowledge	SRS document, use cases, testing and monitoring the code	Refine needs; testing; monitoring

3. Distributed cognition

Distributed cognition was introduced to deter the notion of cognition being only mental states within an individual(Hutchins, 2000; Hutchins & Lintern, 1996). Hutchins argued that these types of assumptions about cognition will limit the researchers in understanding the complex systems(Hutchins & Klausen, 1996; Hutchins & Lintern, 1996). Hence, he introduced the term 'distributed cognition' (DCog) referring to the cognition that is deeply distributed in the social systems and artifacts. DCog has been widely being used by researchers for understanding complex systems like airline and navigation systems (Hutchins & Klausen, 1996; Hutchins & Lintern, 1996); peer tutoring(King, 1998); interdisciplinary teamwork (Derry, DuRussel, & O'Donnell, 1998); classroom practices (Hewitt & Scardamalia, 1998); clinical encounters (Lebeau, 1998); distributed cognitive tasks (Zhang & Norman, 1994) and human computer interaction (Hollan, Hutchins, & Kirsh, 2000; Wright, Fields, & Harrison, 2000).

Distributed cognition primarily exists in three forms of distributed cognitive processes: within social groups; internal and external representational structures; and over time(S. W. Hansen; Hutchins & Lintern, 1996). For our purposes, we will be focusing on the three forms of distributed cognitive processes which are distributed socially; structurally and temporally. In the first form of distribution namely social, knowledge or thought from an individual mind gets distributed and traverses across the human minds. For instance in the case of classroom learning, the tutor and tutee mutually appropriate in building the knowledge in their individual minds (King, 1998). In this type of settings, the students mutually engage in transactive cognitive partnerships. However if we take broader social systems like multidisciplinary teams, the transactive cognitive partnerships will be multi-dimensional. This is because the mutual appropriation of knowledge exists in many to one mapping rather than a simple one to one mapping (Derry, et al., 1998).

In the social distribution of cognition, knowledge holds the key in understanding the different cognitive process. Perkins was the first who classified knowledge in distributed environments into two levels: “content-level” knowledge – the knowledge that deals with facts and procedures; “higher-order” knowledge–the knowledge that deals with problem-solving strategies and justification(Perkins, 1993).

“Content-level” knowledge is a broader term which encompasses both declarative (“knowing what”) and procedural knowledge (“knowing how”). Some of the researchers later parted ways in defining terms like “domain knowledge” (Alexander & Judy, 1988); “domain specific knowledge”(McCutchen, 1986) ;”content-specific knowledge”

(Carpenter, Fennema, Peterson, & Carey, 1988) to indicate the content knowledge that is specific to one specialization or domain.

In the area of information systems (IS) the term domain knowledge is widely being accepted. The domain knowledge in IS discipline has been further divided to IS domain knowledge ("knowing what") and application domain knowledge ("knowing how')(Khatri, Vessey, Ramesh, Clay, & Park, 2006). But in the case of distributed cognitive environment, it is more appropriate to use broad terms like "domain knowledge" for capturing the groups' specific domain knowledge.

"Higher-order" knowledge refers to the computational skills; justification and explanation of the domain concepts (Perkins, 1993). This term has been later called "strategic knowledge" or "application knowledge" to represent task-limited and across-domain strategic knowledge (Pressley, Goodchild, Fleet, Zajchowski, & Evans, 1989) in classroom settings. Application knowledge or strategic knowledge in essence refers to special forms of procedural knowledge ("know how") with high degree of variance from the actual procedures(Alexander & Judy, 1988).

Decision making is another key cognitive activity which mediates the domain knowledge and application knowledge. This activity involves choosing the best possible alternative in a complex situation. In the structural distribution of cognition, human mind and artifacts play a key role in defining the internal and external representational of the structures. For a long time psychologists believed that the internal representations in human mind are made of images(Kosslyn & Pomerantz, 1977; Pylyshyn, 1973). This has been later debated by researchers to include other forms of representations like propositions; data structures; procedures and productions (Pylyshyn, 1973); neural networks (Zhang, 1997). Even though it is not clear what exactly goes in the human mind, there exist different forms of internal representations across human minds which modify or create new ideas. The external representation consists of knowledge and structures in the external environment (Zhang, 1997). These external representations often serve as memory aids; extended memories; archives (Zhang, 2001) or anchor the cognitive behavior (Zhang & Norman, 1994).In addition, in some tasks these external representations act as intrinsic components(Zhang, 2001).

In temporal distribution, the cognitive processes or events of the past influence the future events. Researchers have used multiple time frames like physical time, cultural-historical time used to evaluate the distribution of the cognitive process over time (Salomon, 1997). In short term temporal distribution, interactions of cognitive process takes place primarily between people and artifacts. But, in long term distribution, the events of the past also influence the cognitive process of the present.

The temporal distribution is an emergent property of the system and can be found in activities of transactive memory systems (Moore & Rocklin, 1998; Wegner, 1987). Transactive memory is a systems concept developed to understand how the group process and organizes the information. This concept is evident in the activities like encoding and retrieving. In the process of transactive encoding a group encodes information which often involves complex negotiations. On the other hand, transactive retrieval activity involves determining the location of information coming from multiple locations and memory systems (Wegner, 1987).

3.1. Distribution in open source projects

In an open source community, the RE knowledge gets interspersed across different dimensions of distribution like social, structural and temporal. Even the origins of the evolution of requirements come from diverse sources. The literature in the past has identified that the requirements evolve due to the impact of five different sources: developers, users, explicit and implicit standards or building prototypes (Massey, 2002). The first two sources represent a need faced by individuals. The next two sources arise for meeting industry standards. The last source represents a learning process which acts as a triggering point for creating new requirements. The huge distribution in requirement evolution and RE dimensions makes it hard to understand how they manage the quality in requirements.

RE tasks in a traditional software development involve elicitation, specification, negotiation, verification and validation vis-à-vis discovery, specification, negotiation, prioritization and monitoring(Christel, Kang, & INST., 1992; S. Hansen, Berente, & Lyytinen, 2009). Scacchi argues that open source projects don't follow the "logic based requirement notations" or "formalisms" and hence he refers to the RE practices in open source as being informal(Scacchi, 2002). The informal ways of collecting requirements a.k.a. software informalisms paved way for synonymous RE task structures in open source i.e. 1) "Assertion" for elicitation 2)" Reading, sense-making, accountability" for analysis 3) "Continually emerging webs of software discourse" for requirements specification and modeling 4) "Condensing discourse" for requirements validation 5) "global access to open software webs" for communicating requirements(Scacchi, 2002). But the open source RE task structure just described above is not a generalized frame work and hence cannot be used to analyze different software development projects. For generalizing the findings across different software development methodologies a common frame work is needed and hence we chose the framer work of discover, specify, negotiate & prioritize, monitor(S. Hansen, et al., 2009). The generic RE tasks encompass requirements knowledge distributed across social, structural and temporal dimensions which eventually impact the quality of the requirements. Hence we will be discussing about the social, structural and temporal distribution in open source projects.

3.1.1. Social distribution

Social distribution refers to the distribution of social actors among the projects. This distribution can be observed in people's skills, roles and knowledge. The skills set of these social actors is crucial in understanding how the requirement knowledge gets populated in these communities. The skills and knowledge that we are referring here are the requirements engineering skills like interpersonal and technical(Nuseibeh & Easterbrook, 2000). It can either be domain or strategic knowledge(Perkins, 1993). The interpersonal skills here refer to the ability to negotiate, communicate and articulate the requirements. The technical skills refer to the mastery of skills in that specific field of software that is being developed. The skillfulness can help in producing thoughtful insights on the feasibility of the requirements which can be strategic knowledge(Nuseibeh & Easterbrook, 2000).

Open source doesn't have a clear hierarchical structure which makes it difficult to understand how the skill sets vary among different groups. The past literature has observed that the structure of the social distribution in open source resembles the shape of an onion (Crowston & Howison, 2005).The core of the onion is formed by the core developers and the concentric layers around the onion are co-developers, active developers and passive developers, respectively(see Fig.1). The activity and involvement of these developers decreases as they move away from the core(Crowston & Howison, 2005). The active participation of the core and active developers reciprocates the tacit and explicit knowledge of the core members to other developers (see Fig.1. which shows the accumulation of RE knowledge). In addition, core developers actively provide guidance to other developers by sharing the archival knowledge present in the community forums, email threads etc(Sowe, Stamelos, & Angelis, 2008) .To emphasize the knowledge sharing and distribution across developers, Sowe reclassified developers/users into knowledge seekers and knowledge providers(Sowe, et al., 2008) . This is a simplistic model which considers knowledge to be shared among these two types of members of the community: developers and users. Members often change the role of knowledge seeker (user) and knowledge provider (developer) in the development process.

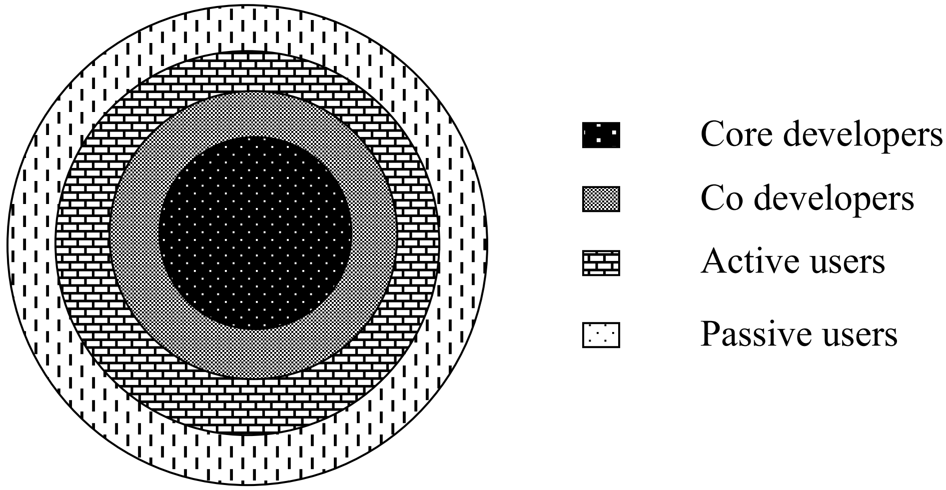

Figure 1. RE Knowledge distribution across different actors (adopted from Crowston, 2005)

3.1.2. Structural distribution

In a traditional development setting, stakeholders use software requirements specification (SRS) documents as a central artifact for storing the requirements. However in open source RE knowledge is formed across multiple artifacts like forums, emails, chats and bulletin boards. Moreover, these communities don't have formal documentation(Scacchi, 2002) which make it challenging to understand how there exists a unison in RE knowledge.

One form of representation of knowledge can be found in the internal minds of the developer. The process of internalization of knowledge among humans is often debated among psychologists. Three central theories have been proposed to understand how a human mind internalizes the knowledge: Experience-Centered (EC); Interactive and Mind-centered (MC). It is important to note that the internal representations of the knowledge go back and forth between the poles of experience-centered to mind-centered approaches. Hence, the process of computation of requirements happens in a continuum of representational structures in the human mind. These representations structures usually take the form of images, propositions, schemas etc. at different levels based on cognition, culture and environment(Reynolds, Sinatra, & Jetton, 1996). The internal representations in developers' mind in traditional and open source vary depending on the amount of time spent on formulating or refining requirements. In traditional software development, the number of internal representations will be finite but in open source they can be huge depending on the size and participations of the members of the community.

The second form of structural representation can be found in artifacts and ecology i.e., external representations. The external representations are the places where the tangible requirement knowledge can be seen. The external representations of requirements can be found in forums, threads, emails, chats and other documents. Some open source projects use tools like Concurrent version system (CVS) for maintaining the history and documents (Amant & Still, 2007) for capturing the RE knowledge. RE knowledge in open source is captured in artifacts pertaining to technical, organizational or institutional knowledge (see Fig.2). Artifacts like source code, versions of the code contain RE technical knowledge in which implicit knowledge of requirement is stored. Email threads, chats, discussion threads contain RE organizational knowledge relating to allocation of activities, determination of requirements, revisions of requirements etc. Licensing agreements contain implicit RE institutional knowledge relating to the use of code for commercial or licensing purposes.

The documentations of requirements are usually quite informal. In some cases, developers provide documentation for further revisions of the code. This kind of documentation usually tells the new developers on how to use and revise the code. However, documentation is usually lagging behind as it is written after the code is being generated. In case of large open source projects, the documentation is often done by a separate documentation team. These kinds of documentation procedures help the developers in understanding the requirements for the future enhancements of the code(Amant & Still, 2007).

Figure 2. RE Knowledge distribution across different artifacts (adopted from Lanzara, 2003)

3.1.3. Temporal distribution

Temporal distribution refers to the cognitive events of past influencing the future events. In short term temporal distribution, the RE knowledge gets determined by the interaction of people with artifacts. The interactions in open source are mainly 'virtual' and are mediated by computer. As two or more computers are used in this type of communication, we can refer this as Computer Mediated Communication (CMC)(Walther, 1996). CMC anchors the transfer of information to the members of open source community through internet by forums, emails, chats etc. The interactions in forum/email thread follow a “self-sustaining process”(Lanzara & Morner, 2003) wherein the “threads may emerge unexpectedly, then suddenly disappear”(Lanzara & Morner, 2003). In addition, the forum/email threads can have ‘flocking effect’ where in the attention of the participants is suddenly diverted towards an emerging theme/issue. The forum/email thread patterns “stimulate reciprocity and become the basis for coordination and knowledge making”(Lanzara & Morner, 2003). Hence, the sequential threads generated by members indicate the accumulation of RE knowledge

The information or knowledge on RE is encoded in the forum threads. In transactive encoding, members discuss the RE knowledge and determine what type of information is important. Transactive encoding activities can vary from a simple encoding of information to “complex negotiations”; allocation of responsibilities; storage of information. At the same time, transactive retrieval activities involve retrieval of information from multiple sources. The retrieval process can either come directly from one source or may involve interplay of different sources(Wegner, 1987).

In long term temporal distribution, past events, environment and other factors constrain the future set of requirements. Transactive encoding in long term temporal distribution involves storage and negotiations of the RE knowledge. Once the RE knowledge is encoded, it will drive the future set of requirements by acting as constraints for future requirements. In addition, transactive retrieval activities involves retrieval of the archived RE knowledge for formulating new requirements(Wegner, 1987).

4. PROPOSITIONS

In this section, we will be using distributed cognition lens for developing propositions for the quality of requirements in open source. Specifically, we will be using the constructs that we discussed earlier in social, structural, and temporal distribution aspects in developing these propositions.

4.1. Social distribution and quality of requirements

The phenomenon of social distribution of cognition is prevalent in the open source communities. People in these communities come from diverse set of professions, cultures, age groups and education levels (Ghosh, Glott, Krieger, & Robles, 2002). However, these communities have certain norms which restrict only qualified individuals for participating in the community (Von Krogh, Spaeth, & Lakhani, 2003). This kind of "restricted" diversity allows the community to have less variation in terms of cognition – domain knowledge, decision making, and application knowledge.

For instance, in the case of a Chandra X-ray Center Data System (CXCDS), developers had to build software applications for analyzing remote sensed data. Most of the community members in this community had strong educational backgrounds in astrophysics and software development and were not just mere software professionals. As the developers had knowledge in both domains, lesser time is spent on recapitulating the basics of the astrophysics or software development. It can be seen that minimalistic variations in domain knowledge not only allowed the developers in spending more time on RE activities but also helped them in refining the requirements. As the requirements were very complex, developers had to spend more time in clarification to find out the hidden requirements. Hence, we posit that lesser variations in domain knowledge allows the developers to spend more time on understanding the requirements helping them in improve the quality of the requirements (Scacchi, 2002).

> *Proposition 1a. The fewer the variations in domain knowledge across developers, the more time spent on RE activities (discovery, analyze, specify& negotiate, monitor) for clarification and refinement of the requirements enhancing a higher quality in requirements.*

The social distribution is also found in the roles of the developers. People in the open source communities get roles based on the amount of participation and contribution. Core developers form the central part of the community and are responsible for requirements discovery, monitoring, decision making and code development. Decision making is an integral activity carried out by the core developers which involves complex cognitive computations for determining the best possible alternative in a given situation. An example of Apache demonstrates that they follow voting procedure for determining the best possible alternative among the requirements (Fielding, 1999). The people in the core developers gain their status after years of experience and contribution to the community.

Hence the presence of the domain and application knowledge helps them in making sure that the requirements are feasible. Hence, we posit that

> *Proposition 1b. The decision making rights among core developers increases the quality of the requirements.*

The social distribution of cognition can vary based on how one applies the knowledge. This kind of knowledge is commonly referred to as application or strategic knowledge. We will be using the term 'application knowledge' to understand its implications on the quality of the requirements. This knowledge is quite essential as it can transform the domain knowledge to application domain by a recurring inquiry. The recurring inquiry helps in clarifying and condensing the requirements. For instance, in specification phase, the requirements are specified by condensation of the communication messages. The process of condensation or specification takes place through a computer mediated dialectic process. The higher order reasoning skills among developers calls for clarification among the requirements. The egalitarian structure of the open source allows for transparency and hence responds to every clarification promptly. This type of cyclical inquiry helps in reducing the ambiguous nature of the requirements. Hence we posit,

> *Proposition 1c. The higher the application knowledge in developers, the more time spent on RE activities (analyze, specify& negotiate) for condensing the requirements leading to a higher quality in requirements.*

More specifically, developers like active developers are involved in the preparation of use cases (Crowston & Howison, 2005). Hence, the application knowledge of the developers is crucial in forming high quality use cases for the requirements. The use cases usually help the developers in a clear understanding of the requirements. Hence, we posit that

> *Proposition 1d. The higher the application knowledge of active developers, the higher will be the quality of the use cases which in turn will enhance a higher quality in requirements.*

It can be noted that the social distribution of cognition can be found in terms of presence of domain knowledge, decision making and the application of the knowledge. All these activities seem to have a significant effect in impacting the quality of the requirements in open source projects.

4.2. Structural distribution and quality of requirements

Structural distribution here refers to the set of cognitive process that deals with the distribution of cognition between internal and external representational structures(Hutchins & Lintern, 1996). The internal representations of the knowledge are crucial in forming new forms of knowledge. These internal representations exist in variety of forms like images, propositions, schemas, neural networks etc(Zhang & Norman, 1994). It is important to note that the internal representations are progressive representations which are influenced by the internal factors like domain knowledge and external factors like artifacts and environment. However, it is not clear how these developers' form images or schemas which in turn lead to world-class software applications.

In discovery phase, internal representations are crucial as they might evolve from developers' mind (Raymond, 2001). As requirements are essentially future visions for building a software system, internal representations in the human mind are critical in their formation and refinement. The collective process of voting on future requirements by the core developers helps them in reaching a consensus on the future set of requirements. In order to perform this activity, there are a series of internal representational structures which gets formed in developers' mind which can give clarity and coherence to the requirements. Hence we posit that,

> *Proposition 2a. The higher the internal representations, the lesser the ambiguities and richer will be the quality of the requirements.*

External representations are crucial in open source as they can take the role of external memory. To support, we witnessed the profound impacts of external representation in the fields of learning(Zhang, 1997). In the case of open source, external representations include forums, chats, bboards, emails and offline chats.

In RE phases like specification, negotiation, prioritization and monitoring, external representation bolster most of the computational processes. Developers debate in these external representational structures to give more clarity to the requirements. Hence, we posit

> *Proposition 2b. The higher the usage of external representations like forums and emails the clearer will be the requirements promoting a higher quality in requirements.*

Some open source projects use external representation tools like Concurrent version system(CVS) for maintaining the history and documents(Amant & Still, 2007). This kind of documentation process helps the co-developers in clearly understanding the embedded requirements. Hence we posit that

> *Proposition 2c. The higher the usage of external tools like CVS the clearer will be the requirements promoting a higher quality in requirements.*

4.3. Temporal distribution and quality of requirements

Temporal distribution refers to the distribution of cognitive processes over time. In this type of distribution, past events, interactions influence the future set of cognitive events. In most of the cases the distributions can be short term and long term. Short term temporal distribution here refers to the distribution of events that happen over a short term period. Especially in the case of open source, the requirements get transformed instantaneously because of the constant inquiry process in the form of email threads, forums and discussion boards. The transactive encoding and retrieving plays a key role in interactions of the people and artifacts which in turn helps in refining the requirements (Wegner, 1987). Hence we posit that,

Proposition 3a. The higher the time spent on short term transactive encoding and retrieving activities, the higher will be the time spent on refining the requirements enhancing a higher quality in requirements.

In long term transactional activities, the experiences of the people, artifacts play a great role in clearly articulating the requirements. For instance, in the case of developing an email popup client, Eric Raymond, one of the developers searched for the existing open source software for satisfying his own personal requirements. He then takes up one of the existing pop client and refines it in order to suit his own personal needs (Raymond, 2001). This is a typical case of embedded requirements driving the future set of requirements. In this case, long term transactional activities like encoding and retrieving helped in a clear articulation of the future set of requirements (Wegner, 1987). Hence we posit that,

Proposition 3b. The higher the time spent on long term transactive encoding and retrieving activities the higher will be the time spent on refining the requirements enhancing a higher quality in requirements.

Open source projects are unique as they engage the developers in different forms of interaction in discovering, specifying, analyzing and monitoring the requirements. The process of engagement is mostly computer mediated which allows the developers a time lag for reacting to the set of the questions posed by other developers. This engagement process allows the developers in storing, cross referencing the requirements(Sowe, et al., 2008). In this way, lesser time is spent on monitoring the requirements and more time on clarifying the requirements through threaded messages. Hence this computer-mediated process of engaging developers promotes in improving the quality of the requirements.

Proposition 3c. The computer mediated process of engaging developers' helps them in spending less time in monitoring requirements and more time in clarifying requirements through threaded messages promoting a overall improvement in the quality of the requirements.

5.RESEARCH DESIGN

The distribution cognition lens requires a deeper understanding of the nature of the existing system. Past studies of this theory have used extensive ethnographical methods for investigating airline and navigation systems(Hutchins & Klausen, 1996; Hutchins & Lintern, 1996) . However, the distributed nature of the open source calls for special investigation techniques to study the system. Hence we will be conducting interviews in addition to the content analysis for getting a better understanding of the requirements in open source (Silverman, 2005).

In order to study the quality of requirements in open source development projects, as a preliminary step we will perform content analysis using the qualitative data from forums, discussion boards and email threads. The propositions that we have observed from the previous literature require a rigorous investigation of social, structural and temporal aspects of open source requirements. Hence, we developed coding schema from the existing literature. To further illustrate the coding schema we have investigated the user forums of Adobe (see Table). We will be using the same coding schema to study open source projects of Mozilla and Apache.

Table 3: Coding schema for studying open source projects

Category		Definition	Examples
Social distribution	1.Domain knowledge	Knowledge associated with facts and procedures of a specific topic(Perkins, 1993).	*"Open Office 3 seems to replace it by a hypen (which is greyed), and InDesign CS4 also seems to replace the character and does no word-wrap at this position".*
	2.Application knowledge	Knowledge associated with problem-solving skills, reasoning, and justification on a specific topic(Perkins, 1993).	*"Word and InDesign do use the hyphen glyph when a non-breaking hyphen is undefined …I think this is a reasonable feature request either for TLF or the underlying flash.text.engine that powers it."*
	3.Decision making	Ability and authority to make decisions on the topic(Fielding, 1999).	*"Users will have to upgrade. But folks usually do this pretty quickly after a release. - Forum user1"*
Structural distribution	1.Internal representation	Representations which can be in the form of propositions, images and data structures(Pylyshyn, 1973).	"*Of course it would be very convenient if Flash could handle it the same way*"
	2.External representation	Representations which act as memory aids or archives(Zhang,	*"You will find it at: http://labs.adobe.com/technologies/flashplayer10/"*

		2001).	
Temporal distribution	1.Short term transactive encoding, retrieving	Interaction between people and artifacts for encoding/retrieving RE knowledge(Wegner, 1987).	*"When I paste your markup into Notepad, I see the same box for the hyphen that I see when viewing it in the TLF demo editor. Are you saying that you see the hyphen display in other Windows apps but not TLF"*
	2.Long term transactive encoding, retrieving	Interaction of the people/artifacts to retrieve/encode archival RE knowledge(Wegner, 1987).	*"Many Windows applications do not do any kind of substitution whatsoever (Notepad is one example)"*
	3.Computer mediation	Interaction mediated by computers instead of face-to-face(Walther, 1996).	*"I can't promise anything, but I'll take this request back to the team.* *Thanks!* *-Forum user2"*
Quality	1.Consistent	One possible interpretation(B. W. Boehm, 1984).	*"you're right, Times New Roman does not contain the glyph."*
	2.Complete	Present and fully developed(B. W. Boehm, 1984).	*"The non-breaking hyphen (\u2011, &#8209) is not displayed correctly - the wrong glyph is shown.* *To reproduce, do the following:* *1. Start the TLF demo editor http://labs.adobe.com/technologies/textlayout/demos/* *2. Import the markup below* *Results:* *- Line wrapping is correct: no line break at the hyphen* *- Times New Roman on Windows does have the glyph defined – it should look like an ordinary hyphen.* *This was seen in Build 3291.* *Cheers* *Forum user3"*
	3.Feasible	Met in real time without exceeding costs(B. W. Boehm, 1984).	*"Is there any effort going toward getting it working in 10.0 or will we just need to have all of our users upgrade to 10.1 when it is officially released?"*
	4.Testable	Can be examined precisely(B. W. Boehm, 1984).	*"The non-breaking hyphen (\u2011, &#8209) is not displayed correctly"*

(Source: http://forums.adobe.com/thread/29480)

Qualitative data is essential in understanding the internal process and for providing a broader understanding of the underlying reality(Strauss, Corbin, & Lynch, 1990). The subjective nature of qualitative study can provide richer cases for evaluating the theoretical propositions. We would be using a structured case approach to investigate the current situation by unfolding the existing literature(Dawson, 2008; Eisenhardt, 1989).

6. Discussion, conclusions and limitations

A review of the existing literature on requirements engineering has indicated many avenues of growth for designing future software systems. For over past 50 years we have been following a reductionist approach for analyzing the requirements which in a way has limited our understanding of the requirement engineering process(Curtis, Krasner, & Iscoe, 1988). Despite a wide scale research in RE, the root causes for the RE issues are yet to be revealed. One of the burgeoning issues in RE occurs because of lack of quality in the requirements. Researchers in the past have used the qualitative and quantitative approaches for improving the quality of the requirements(Mylopoulos, et al., 1992; Robinson & Fickas, 1994) & quantitative(Keller, et al., 1990). However, these methods had great limitations as they ignore some of the crucial aspects of the social, structural and temporal aspects of the project and organization.

In the recent years, open source projects have shown a great amount of success in handling the requirements. Even though they don’t have any formal RE process they have been able to deliver world class sofwares without falling into the traps of RE issues. The process of gathering requirements in open source revolves around the social, structural and temporal aspects of the cognition. Hence, we used distributed cognition for understanding their ability in producing high quality requirements.

Distributed cognition lens understands systems by identifying the three cognitive processes involved in social, structural and temporal distribution(Hutchins & Lintern, 1996). The cognitive processes in social distribution involve sharing of the knowledge and skills across team members coming from various professions, cultures, age groups and education levels(Ghosh, et al., 2002). The past studies on distributed cognition tend to ignore the representational states present across social, structural and temporal domains. Our study provides a rigorous technique to investigate distributed cognition by looking at the both the “system” as well the individual constructs.

Over the past few decades, revolutionary methodologies such as open source have been able to produce world class sofwares. Even with the lack of formal documentation they are capable of maintaining high quality in requirements through different forms of knowledge structures, external representations and negotiation processes.

The current study has practical implications to the open source community for understanding the role of different constructs in formulating the requirements. For instance the degree of commonality across developers helps in spending less time on requirement clarification. Hence the owners of the multi-disciplinary open source projects can compose their teams based on the similarities in domain knowledge of the developers. At the same time, interaction in forums or threaded emails can stimulate active validation of the requirements on a continuum basis. Usage of external representations in storing the requirements can help active/passive developers in understanding requirement activities in a better way. External representations also help core developers in actively addressing the RE issues raised by other developers by cross-referencing the email or forum threads. Two critical managerial implications from this study are that 1) requirements documents should not be thought of as history archives but as engaging documents and 2) RE activities should not be viewed as a one-time activity but as an iterative engagement activity. In addition, the current study extends the body of literature on quality of requirements in open source. Currently there exist no studies specifically on the quality of the requirements in open source(Aksulu & Wade).

As the open source teams are globally distributed it is hard to capture the cognitive process embedded in the spatial settings. Hence, one set of limitations of the study is its inability to capture the cognitive process in the spatial domain. In addition, the current study doesn't account for project size, scope and trust among the developers. Further the theoretical propositions lack empirical evidence. Hence the future research should focus on validating the propositions and understanding the quality issues in open source requirements.

REFERENCES

Aksulu, A., & Wade, M. R. A Comprehensive Review and Synthesis of Open Source Research. *Journal of the Association for Information Systems, 11*(11), 6.

Alexander, P., & Judy, J. (1988). The interaction of domain-specific and strategic knowledge in academic performance. *Review of Educational Research, 58*(4), 375.

Amant, K., & Still, B. (2007). *Handbook of research on open source software: technological, economic, and social perspectives*: Information Science Publishing.

Bell, T., & Thayer, T. (1976). *Software requirements: Are they really a problem?*

Bergman, M., King, J. L., & Lyytinen, K. (2002). Large-scale requirements analysis revisited: The need for understanding the political ecology of requirements engineering. *Requirements Engineering, 7*(3), 152-171.

Boehm, B. (1983). *The economics of software maintenance*.

Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. *Software, IEEE, 1*(1), 75-88.

Carpenter, T., Fennema, E., Peterson, P., & Carey, D. (1988). Teachers' pedagogical content knowledge of students' problem solving in elementary arithmetic. *Journal for Research in Mathematics Education, 19*(5), 385-401.

Cheng, B., & Atlee, J. (2007). *Research directions in requirements engineering*.

Cheng, B. H. C., & Atlee, J. M. (2007). *Research Directions in Requirements Engineering*. Paper presented at the 2007 Future of Software Engineering.

Christel, M. G., Kang, K. C., & INST., C.-M. U. P. P. S. E. (1992). *Issues in requirements elicitation*: Citeseer.

Chung, L., & do Prado Leite, J. (2009). On non-functional requirements in software engineering. *Conceptual Modeling: Foundations and Applications*, 363-379.

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday, 10*(2), 1–100.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31*(11), 1268-1287.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., et al. (1993). *Identifying and measuring quality in a software requirements specification*.

Dawson, L. (2008). Active Exploration of Emerging Themes in a Study of Object-Oriented Requirements Engineering: The "Evolutionary Case" Approach. *Electronic Journal of Business Research Methods, 6*(1), 29-42.

Derry, S., DuRussel, L., & O'Donnell, A. (1998). Individual and distributed cognitions in interdisciplinary teamwork: A developing case study and emerging theory. *Educational Psychology Review, 10*(1), 25-56.

Eisenhardt, K. (1989). Building theories from case study research. *Academy of management review, 14*(4), 532-550.

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

Ghosh, R., Glott, R., Krieger, B., & Robles, G. (2002). Free/libre and open source software: Survey and study: Maastricht Economic Research Institute on Innovation and Technology, University of Maastricht, The Netherlands, June.

Hansen, S., Berente, N., & Lyytinen, K. (2009). Requirements in the 21st century: Current practice and emerging trends. *Design Requirements Engineering: A Ten-Year Perspective*, 44-87.

Hansen, S. W. *A Socio-Technical Perspective on Requirements Engineering.* CASE WESTERN RESERVE UNIVERSITY.

Hewitt, J., & Scardamalia, M. (1998). Design principles for distributed knowledge building processes. *Educational Psychology Review, 10*(1), 75-96.

Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction (TOCHI), 7*(2), 196.

Hutchins, E. (2000). Distributed cognition. *Internacional Enciclopedia of the Social and Behavioral Sciences*.

Hutchins, E., & Klausen, T. (1996). Distributed cognition in an airline cockpit. *Cognition and communication at work*, 15–34.

Hutchins, E., & Lintern, G. (1996). *Cognition in the Wild*: MIT press Cambridge, MA.

Jirotka, M., & Goguen, J. (1994). *Requirements engineering: social and technical issues*: Academic Press Professional, Inc. San Diego, CA, USA.

Keller, S., Kahn, L., & Panara, R. (1990). Specifying software quality requirements with metrics. *System and Software Requirements Engineering*, 145–163.

Khatri, V., Vessey, I., Ramesh, V., Clay, P., & Park, S. (2006). Understanding conceptual schemas: Exploring the role of application and IS domain knowledge. *Information Systems Research, 17*(1), 81.

King, A. (1998). Transactive peer tutoring: Distributing cognition and metacognition. *Educational Psychology Review, 10*(1), 57-74.

Kogut, B., & Metiu, A. (2001). Open Source Software Development and Distributed Innovation. *Oxford Review of Economic Policy, 17*(2), 248.

Kosslyn, S., & Pomerantz, J. (1977). Imagery, propositions, and the form of internal representations* 1. *Cognitive Psychology, 9*(1), 52-76.

Lanzara, G., & Morner, M. (2003). *The knowledge ecology of open-source software projects*.

Lebeau, R. (1998). Cognitive tools in a clinical encounter in medicine: Supporting empathy and expertise in distributed systems. *Educational Psychology Review, 10*(1), 3-24.

Liu, X. (1998). A quantitative approach for assessing the priorities of software quality requirements. *The Journal of Systems & Software, 42*(2), 105-113.

Lutz, R. (2002). *Analyzing software requirements errors in safety-critical, embedded systems*.

Massey, B. (2002). *Where do open source requirements come from (and what should we do about it)*.

McCutchen, D. (1986). Domain knowledge and linguistic knowledge in the development of writing ability* 1. *Journal of Memory and Language, 25*(4), 431-444.

Moody, G. (2001). *Rebel code: Inside Linux and the open source revolution*: Perseus Books Group.

Moore, J., & Rocklin, T. (1998). The distribution of distributed cognition: Multiple interpretations and uses. *Educational Psychology Review, 10*(1), 97-113.

Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Transactions on Software Engineering, 18*(6), 483-497.

Nardi, B. (1996). Studying context: A comparison of activity theory, situated action models, and distributed cognition. *Context and consciousness: Activity theory and human-computer interaction*, 69-102.

Nuseibeh, B., & Easterbrook, S. (2000). *Requirements engineering: a roadmap*.

Perkins, D. (1993). Person-plus: A distributed view of thinking and learning. *Distributed cognitions: Psychological and educational considerations*, 88-110.

Pohl, K. (1993). *The three dimensions of requirements engineering*.

Pressley, M., Goodchild, F., Fleet, J., Zajchowski, R., & Evans, E. (1989). The challenges of classroom strategy instruction. *The Elementary School Journal, 89*(3), 301-342.

Pylyshyn, Z. (1973). What the mind's eye tells the mind's brain: A critique of mental imagery. *Psychological bulletin, 80*(1), 1-24.

Raymond, E. (2001). *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*: O'Reilly & Associates, Inc. Sebastopol, CA, USA.

Reynolds, R., Sinatra, G., & Jetton, T. (1996). Views of knowledge acquisition and representation: A continuum from experience centered to mind. *Educational Psychologist, 31*(2), 93-104.

Robinson, W., & Fickas, S. (1994). *Supporting multi-perspective requirements engineering.*

Rogers, Y., & Ellis, J. (1994). Distributed cognition: an alternative framework for analysing and explaining collaborative working. *Journal of information technology, 9*, 119-119.

Roman, G. (2006). A taxonomy of current issues in requirements engineering. *Computer, 18*(4), 14-23.

Salomon, G. (1997). *Distributed cognitions: Psychological and educational considerations*: Cambridge Univ Pr.

Scacchi, W. (2002). Understanding the requirements for developing open source software systems. *IEE Proceedings-Software, 149*(1), 24-39.

Silverman, D. (2005). *Doing qualitative research: A practical handbook*: Sage Publications Ltd.

Sommerville, I., & Ransom, J. (2005). An empirical study of industrial requirements engineering process assessment and improvement. *ACM Transactions on Software Engineering and Methodology (TOSEM), 14*(1), 85-117.

Sommerville, I., & Sawyer, P. (1997). *Requirements engineering*: Wiley.

Sowe, S., Stamelos, I., & Angelis, L. (2008). Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software, 81*(3), 431-446.

Strauss, A., Corbin, J., & Lynch, M. (1990). *Basics of qualitative research: Grounded theory procedures and techniques* (Vol. 270): Sage Newbury Park, CA.

Von Hippel, E., & Von Krogh, G. (2003). Open source software and the" private-collective" innovation model: Issues for organization science. *Organization Science*, 209-223.

Von Krogh, G., Spaeth, S., & Lakhani, K. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research policy, 32*(7), 1217-1241.

Walther, J. B. (1996). Computer-mediated communication. *Communication research, 23*(1), 3.

Weber, S. (2004). *The success of open source*: Harvard Univ Pr.

Wegner, D. (1987). Transactive memory: A contemporary analysis of the group mind. *Theories of group behavior*, 185-208.

Wright, P., Fields, R., & Harrison, M. (2000). Analyzing human-computer interaction as distributed cognition: The resources model. *Human-Computer Interaction, 15*(1), 1-41.

Zhang, J. (1997). The nature of external representations in problem solving. *Cognitive science, 21*(2), 179-217.

Zhang, J. (2001). External representations in complex information processing tasks. *Encyclopedia of library and information science, 68*(31), 164-180.

Zhang, J., & Norman, D. (1994). Representations in distributed cognitive tasks* 1. *Cognitive science, 18*(1), 87-122.