Requirements Computation: Analyzing Requirements Cognition in Multiple Development Paradigms

Completed Research Paper

Sean W. Hansen
Saunders College of Business
Rochester Institute of Technology
shansen@saunders.rit.edu

Kalle J. Lyytinen
Weatherhead School of Management
Case Western Reserve University
kalle.lyytinen@case.edu

Abstract

The identification and management of users' requirements have been a persistent challenge confronting software development projects for decades. Complicating this challenge, research on requirements engineering processes has failed to keep pace with significant changes in the lived practice of software development. In this paper, we consider requirements-oriented processes as a socio-technical computational task in which diverse individuals and artifacts collaboratively "compute" the requirements for an envisioned software resource. Through the perspective of distributed cognition, we analyze the distributed requirements activities of three information systems development projects, each representing a distinct development methodology – structured development, agile development, and open source software development. We construct models of the computational structures of these projects to support a novel analytical basis for comparison and contrast of requirements-oriented efforts in these development methodologies.

Keywords

Distributed cognition, Requirements engineering, Structured development, Agile software development, Open source software development.

Introduction

Requirements processes refers to activities by which information systems (IS) designers discover, specify, validate, and manage the functionality that a proposed software platform is expected to possess and constraints to which it must conform (Kotonya and Sommerville 1998). Despite decades of research in the requirements arena, challenges arising from the failure to effectively articulate and address desired requirements have continued to plague IS development projects (Aurum and Wohlin 2005; Cerpa and Verner 2009; Hickey and Davis 2003). The persistence of requirements-based challenges is further complicated by the rapidly changing nature of IS design practices (Jarke et al. 2011). For example, requirements requirements-oriented activities are increasingly distributed across locations, organizations, stakeholder groups, and time (Hansen et al. 2009). Such distribution poses new challenges and presents a significant contrast to the traditional view of the requirements process in which a small group of analysts specifies a system in relative isolation prior to downstream development (Jarke and Pohl 1994). In addition, the software development domain has experienced an explosion of new methods and technologies including the rise of commercial off-the-shelf (COTS) solutions, framework-based development (e.g., SOA), agile development methods, and open source software development (OSSD). Interestingly, some of these novel methods (e.g., agile development) have emerged specifically in reaction to perceived shortcomings of heavyweight requirements engineering approaches employed in traditional methodologies. However, despite these new approaches, the fundamental challenge of determining what user needs a piece of software must meet has persisted (Cao and Ramesh 2008).

In the wake of these changes, research on requirements processes has struggled to stay abreast of practice (Hansen et al. 2009; Jarke et al. 2011). While some research has focused on requirements processes in geographically distributed teams (e.g., Damian et al. 2003; Grünbacher and Braunsberger 2003) and comparisons of requirements activities across different development paradigms (Gacek and Arief 2004; Scacchi 2009), the research literature includes relatively little exploration of the ways in which requirements activities are simultaneously distributed across space, organizations, artifacts, and time. While requirements research maintains a strong design science tradition (e.g., multiple prescribed tools and techniques), the field has been marked by a relative dearth of theoretical foundations and theory development (Jarke et al. 1993). Consequently, we know little about how a change in distribution affects requirements activities and their outcomes (Jarke et al. 2011).

In the current research, we seek to address this gap both empirically and theoretically. Empirically, we use rich field study data to conduct rigorous analyses of task flows in diverse development environments. Theoretically, we develop a cognitive framing of requirements computation based on the principles of distributed cognition. Specifically, we adopt a cognitive perspective which focuses on mechanisms by which participants in the requirements activities process information to arrive at a shared understanding of the needs and constraints of their project stakeholders.

Requirements processes are central to both the problem formulation and solution generation facets of "design cognition" (Carroll 2002; Cross 2001), but they are cognitive processes that are widely distributed. From a cognitive perspective, we frame requirements analysis as a fundamentally socio-technical computational task. That is, a development project is a socio-technical cognitive system, composed of heterogeneous social actors and artifacts which interact to "compute" the requirements for a desired software system (Hansen et al. 2012). In this view, we understand requirements to be computed through various computational structures (i.e., processes for exchange between elements of a cognitive system), which are expected to vary across different development paradigms (e.g., waterfall development vs. OSSD). Further, we argue that requirements-oriented computations can be modeled as variations in cognitive workflows, which, when successful, achieve *requirements closure*— i.e., a state where requirements are viewed as clear, agreed upon, and implementable. Finally, we contend that, in light of the diversity of configurations, computational structures may vary in their effectiveness at achieving such closure. Thus, we seek to address the following key research questions:

- What are the characteristic modes of social, structural, and temporal distribution of requirements activities in different development environments?
- What computational structures are observed in different development environments?
- How do these computational structures differ across methodologies?

To address these questions, we present a multi-site field study of software development projects in three widely-recognized development environments – structured development, agile software development, and OSSD. In analyzing these project environments, we draw upon the Theory of Distributed Cognition (Hollan et al. 2000; Hutchins 1995a) to identify and model the characteristic modes of cognitive distribution within these complex work environments.

Requirements Engineering

Whether or not they are explicitly acknowledged by designers, requirements processes represent an essential facet of any software development project. Specifically, in any design effort certain foundational questions must be addressed: What do we want to create? What features and functionality should the new system possess? What practical objectives are guiding the design effort? Frameworks within the research literature have divided requirements activities into multiple distinct tasks (Dorfman 1997). In our discussion, we focus on three widely-acknowledged requirements facets: (1) discovery, (2) specification, and (3) validation and verification, as they represent distinct and necessary cognitive outcomes for a successful computation of requirements. Requirements efforts must address the ways in which software development teams manage and coordinate these three activities to achieve desired outcomes.

- *Discovery*: The process by which designers identify the organizational, individual, technological, or other needs that must be met by the envisioned software (Kotonya and Sommerville 1998; Loucopoulos and Karakostas 1995; Maiden 2008; Piller et al. 2004).

- *Specification***:** The rendering of discovered requirements into a representational form, so that the knowledge can be validated for correctness, clarity, feasibility, and coherence, and thereafter used in downstream design activities (Borgida et al. 1985; Mylopoulos 1998; van Lamsweerde 2000). Specification forms the point of transition where articulated needs are extended with functional and technical design implications.
- *Validation & Verification:* The processes used to assess the degree to which requirements processes have been conducted effectively and will support downstream design. The two terms are generally grouped together in the requirements literature. *Validation* is the process of ensuring that requirements accurately reflect the intentions of the stakeholders (Zhu et al. 2002). By contrast, *verification* focuses on the degree to which requirements conform to accepted standards of requirements quality (Wallace and Ippolito 1997) (Boehm 1984; Wiegers 1999). Boehm (1984) captures the distinction succinctly when he states that validation addresses the question "Am I building the right product?"; while verification asks "Am I building the product right?" (p. 75).

From a cognitive perspective, all software design projects must address the overall coordination and integration of the three tasks. However, the identification of these tasks reflects no assumptions of their temporal execution/sequencing, actors involved, means, local goals, or spatial organization. Indeed, as alluded to above, in some development environments (e.g., agile development), project teams might avoid the use of the term "requirements" at all, much less the task labels that we adopt. Nevertheless, we contend that these activities represent *essential* cognitive tasks inherent in any design process.

Distributed Cognition

Distributed cognition is a branch of cognitive science that argues that cognitive processes, such as memory, decision making, and problem-solving, are not limited to the internal mental states of individuals (Hollan et al. 2000; Hutchins 1995a; Hutchins 1995b; Magnus 2007). The development of the theory was motivated by research on teams engaged in complex, collaborative environments, such as naval navigation (Hutchins 1995a), air traffic control (Halverson 1994), and engineering teams (Rogers 1993). In these settings, information processing is not limited to individual actors; rather, it is distributed across members of a group. Furthermore, a significant portion of the cognitive load is borne by the artifacts used by group members.

By framing cognition as "the propagation of representational state across representational media" (Hutchins 1995a: p. 118), distributed cognition extends the unit of analysis for cognitive activity from the individual to the entire group accomplishing to a given task. With this fundamental shift in perspective, the theory develops three critical assertions (Hutchins 2000): 1) cognitive processes are distributed among members of social groups (which we label *social distribution*), 2) cognition employs both internal and external structures (*structural distribution*), and 3) cognitive processes are distributed over time (*temporal distribution*). Table 1 provides a summary of these mechanisms and their applicability to software development in general.

Table 1. Modes of Cognitive Distribution

Modes of Distribution	Description	Example in Software Development
Social Distribution	Cognitive processes distributed across members of a group; individuals play different roles in the information processing and action of the group.	Vast majority of software design employs a team structure (Guinan et al. 1998); Diversity of knowledge required (Levina and Vaast 2005; Walz et al. 1993).
Structural Distribution	Intertwining of internal and external structure in cognitive processes; offloading of cognitive demands onto external artifacts.	Formal modeling of requirements creating external structures to enhance internal human cognition (van Lamsweerde 2000).
Temporal Distribution	Cognitive processes distributed across time; earlier decisions and actions influence cognitive processes enacted later.	Integration of previous designs and components into new systems (Li et al. 2008); Requirements reuse (Cybulski and Reed 2000; Majchrzak et al. 2004)

In addition to the basic structures of cognitive distribution, Hansen et al. (2012) call attention to some dynamic processes that are characteristic of distributed cognitive systems, including redundancy of

knowledge, shared understanding to cognitive transformations, transparency of action, and degree of cognitive offloading. *Knowledge redundancy* refers to the observation that collaborative systems are more robust when there is a degree of redundancy in the knowledge of actors, since the failure of any single element can be compensated for by the knowledge and experiences of other elements of the system (Hutchins 1995a). *Shared understanding* means that effective collaborators must be aware of how each contributes to achieving a system's broader goals. This is similar to idea of 'heedful interrelating' in the concept of collective mind developed by Weick and Roberts (1993). *Transparency of action* implies that a distributed cognitive system functions more effectively when individual members can "see" what the others elements (both human and artificial) are doing (Kirsh 1999). Finally, *cognitive offloading* relates to the degree to which cognitive load is transferred from human actors to artifacts through structural distribution (Larkin and Simon 1987).

Research Design

Multi-Site Field Study

In this study, we conduct a multi-site field study of ongoing systems development projects. This multi-case approach enables us to engage in a rich exploration of the sociotechnical, cognitive process of practicing ISD professionals (Eisenhardt 1989; Yin 2003). The unit of analysis for the study is the individual ISD project. Specifically, our analysis focused on ISD projects employing three distinct development approaches: 1) structured, platform-based development, 2) agile software development, and 3) OSSD. The site inquiries were conducted in accordance with prevailing case study field procedures, including the development of a case study protocol prior to data collection, triangulation using multiple sources of evidence, and the maintenance of a chain of evidence (Yin 2003).

Prior to data collection efforts, we collaboratively developed a case study protocol, which included the research questions guiding the inquiry, the essential design of the study (i.e., multiple case analysis within varied development environments), the bases for case selection, the modes of data collection to be executed (i.e., interviewing, direct observation where applicable, documentary review), and agreed upon data collection procedures. This broader protocol document also included a detailed interview protocol to guide our conversations with respondents. The interview protocol incorporated inquiry into individual and organizational backgrounds, the structure of ISD teams/units, ISD environments and practices, specific techniques used for identifying and capturing design requirements, specific technologies or artifacts used by project members, and perceived challenges to effective requirements determination. Within each of these areas, the protocol incorporated multiple contingent probes for additional discussion. Given the space restrictions of the current paper, we have not attached the protocol document, but it is available to interested researchers upon request.

Summary of cases

The following provides a brief introduction to each of the projects studied. To adhere to assurances of confidentiality, we have developed pseudonyms for the first two organizations:

SIS Project. The SIS Project focused on the acquisition, customization, and implementation of a vendor-sourced Student Information System (SIS) ERP at a mid-sized Midwestern U.S. university. The SIS platform was intended to integrate all student information and student-facing administrative functions across the university's nine schools. The project team employed a structured development method with explicit documentation and discussion of requirements and a thorough review and sign-off process for any proposed changes to the COTS platform. The university engaged multiple consultants to work on the project, including a large team from a firm which specialized in enterprise system implementations within higher education. Other consultants were employed for their specialized technical skills. Finally, the project drew significant requirements knowledge from engagement with a web-based user group, called the Higher Education User Group (HEUG), which comprised of other universities who had been previously working on to the software platform being implemented.

BigD Project. SocialAgg[1] is a social media aggregator that develops software for both internal and external clients. In particular, their software is focused on the advanced analysis of large amounts of web content information (i.e., Big Data). SocialAgg development teams employ an agile methodology that is a variation on Scrum. While they are very committed to the agile development philosophy, they are not dogmatic in their application of the Scrum methodology. The development teams operate on a three-week sprint cycle, conduct stand-up meetings two or three times weekly (rather than daily), and do not use a formal agile task board. The firm relies upon Amazon's cloud services for all data storage and maintains their software in a GitHub repository.[2] The specific project that we studied, labeled BigD, focused on the development of a data framework and query tool that enabled the firm to draw business intelligence for the firm and its clients through algorithmic analysis of massive social media data feeds.

Rubinius Project. Rubinius is an OSSD project focusing on the development of a virtual machine (VM) and related compiler for the Ruby programming language. The project is hosted on GitHub. Rubinius is a partially-sponsored OSSD project. In 2007, two years after the project was initiated, the Engine Yard Company began to sponsor several committers of Rubinius to work fulltime on the project. Engine Yard is one of the largest privately-held firms focused on Ruby on Rails and PHP development. The first fully-functional version of Rubinius was released in 2010 and the project has since been working on a 2.0 release. In keeping with the pattern observed in most OSSD projects (Crowston et al. 2006), Rubinius has a small set of core developers and much larger set of peripheral committers who work on the project on a purely voluntary basis. Not surprisingly, in light of the OSS nature of the project, Rubinius does not employ formalized requirements processes. However, the cognitive tasks associated with requirements determination are accomplished through community discussions and activities mediated through a variety of requirements "informalisms" (Scacchi 2009), including emails communications, developer forums, and internet relay chat (IRC) channels (Gacek and Arief 2004).

Data Collection and Analysis

On initial engagement with each of the projects, we performed an in-depth, semi-structured interview with a key respondent with oversight authority for the relevant project. Key areas of focus in the interviews included identification and analysis of processes employed to capture requirements; the allocation and coordination of requirements tasks among participants; artifacts used to capture, share, monitor, and communicate requirements; the flow of requirements knowledge among project roles and artifacts; and the protocols to maintain requirements priorities over the life of the project. Based on the recommendations of the initial respondents, additional interview respondents were identified. To ensure multiple perspectives on the software development processes employed, we sought participation from development leads and individual developers. Where applicable, we also sought participation of project sponsors or executive stakeholders. All of the interviews were conducted using the aforementioned interview protocol. In all, we interviewed 31 subjects (SIS – 9, SocialAgg – 5, Rubinius – 17), with the typical duration being approximately 60 minutes, although the interviews ranged from 45 to 90 minutes. Our interviews were augmented with a review of artifacts that supported the interaction of project participants (e.g., project documentation, requirements specifications, and communication media). In some cases (e.g., not applicable in the OSSD context), we conducted direct observation of participants to assess how different people work and use requirements knowledge within these projects.

In accordance with a grounded analytical approach, the research team began the coding process concurrently with data collection activities. Specifically, our data coding employed a thematic analysis of the case data (Boyatzis 1998). While the thematic analysis was be conducted in line with key principles of grounded theory methodology (Glaser and Strauss 1967; Strauss and Corbin 1990), such as constant comparison and open, axial, and selective coding, it differed from a pure grounded theory approach in that the analysis was informed by the framework drawn from theories of distributed cognition. All data was coded and parsed into categories to identify important themes in the requirements practices of the projects. The coding was conducted using Dedoose, an online collaborative coding software. The coding processes addressed two distinct axes of analysis: (1) requirements tasks reflected, and (2) modes of distribution. The analysis of requirements tasks focused on the processes through which the projects achieved the ends of discovery, specification, negotiation/prioritization, and monitoring. This analysis led

[1] SocialAgg is a pseudonym used to respect assurances of confidentiality offered to informants.
[2] GitHub is a web-based hosting platform for open source software which is based on the Git version control system.

to the identification of the computation structures, or primary cognitive workflows, of a project. The second analytical axis was the forms of cognitive distribution reflected *within* the workflows and tasks. Thus, we coded the distinct social, structural, and temporal distribution mechanisms employed by project teams. Combining the output of these two analytical lenses, we determined forms of distribution within and across tasks as they evolved through discovery, specification, negotiation, and monitoring.

Findings

In this section, we present two sets of findings: First we discuss the forms of cognitive distribution observed in each of the project contexts. We then articulate the computational structures within which these distribution mechanisms are situated.

Cognitive Distribution

We first sought to determine the characteristic modes of social, structural, and temporal distribution of requirements activities in each of the development environments observed (RQ1). The three projects reveal very different patterns with respect to the ways in which cognitive effort is distributed along the social, structural, and temporal axes. In light of space constraints, we highlight the distinguishing features of the three environments along each of these dimensions in Table 2.

Table 2. Social, Structural, and Temporal Distribution Mechanism Observed

Cognitive Distribution Mechanisms	Projects		
	SIS	***BigD***	***Rubinius***
Social Distribution			
Specialization in roles	***High***: Roles are clearly demarcated *"[The Technical Lead has] one developer that tends to do more of the admissions stuff, one in the student financials, and so on … So he has segregated duties." – SIS Technical Consultant*	***Limited***: Emphasis on broad-based skill sets *"My style is to break [work] down into smaller chunks that can be achieved by no more than two people – one to two people. We call this a one-pizza team." – BigD Project Director*	***Limited***: Developers can self-select specialization, but in practice members have similar skill sets and professional backgrounds.
Communica-tion between collaborators	***Formal and Collective***: Formalized structured walkthrough process *"The walkthroughs were really a good idea. It got people around a table. Some of them were pretty [contentious] … but it needed to happen." – SIS Communications Lead*	***Individualized***: Face-to-face interaction in small units *"The war room helps a lot, especially when talking about [issues]… When you're sitting next to one another, you get a lot of information" – BigD Project Manager*	***Computer-mediated***: All communication through IRC, emails, and community posts *"We were on the IRC channel one day and [the concurrency stuff] kind of came as a dare, like 'I bet you can't add concurrency.'" – Rubinius Developer*
External Insights	***High***: Heavy reliance on consultants and user community *"[One challenge] I dealt with I said, 'Put the question out to the user community. Let's find out what they did with this.' So it's a very useful thing." – SIS Technical Consultant*	***High***: Formal R&D function supporting the team *"[The 'front end' is] a leading edge group that is looking at ideas six to nine months down the line that I could eventually productize." – BigD Project Director*	High: Developer conferences and engagement with other project communities *"RubyConf attracts more experienced developers, so it's a good opportunity to talk to people about a lot of very technical aspects of Ruby and Rubinius." – Rubinius Developer*

Table 2. Social, Structural, and Temporal Distribution Mechanism Observed

Cognitive Distribution Mechanisms	Projects		
	SIS	***BigD***	***Rubinius***
Structural Distribution			
Documentation	***High***: Specifications in a standardized format *"The primary mechanism of communication between my folks and the line-of-business folks is that spec document. Then there [are] other documents that get handed off." – SIS Technical Lead*	***Limited:*** Lightweight; explicit avoidance of formality. *"Sometimes, we create documents, but really it's just doing what makes sense in a given case." – BigD Product Manager*	***Limited***: No standardized documentation; email and IRC 'informalisms' *"I think we had a couple of conference calls maybe, but mostly IRC, and just told people what we were gonna be doing." – Rubinius Developer*
Graphical representation	***Extensive***: Mock-ups and white board drawings; secondary mode of presenting requirements *"I just mock it up in the dev environment and then just sit down with these people for an hour or two and hammer on it." SIS Technical Consultant*	***Extensive***: Mock-ups and white board drawings; primary mode of presenting requirements *"[The requirements are] sort of lightweight – it may include mocks, flows, pictures, etc., and so they just get attached to that." – BigD Project Director*	***Limited*** The ideas outlined in text-based 'informalisms' get converted right into code by individual developers.
System-embedded requirements	***Extensive***: Heavy reliance upon COTS platform being modified *"To what degree can we stick to the package and avoid building over it?" – SIS Technical Lead*	***Limited***: Emphasis on creating entirely new functionality	***Extensive***: heavy reliance on Ruby-based resources *"[RubySpec is] an executable part of Ruby language libraries, and that's what Rubinius tests itself against." – Rubinius Core Developer*
Temporal Distribution			
Iteration	***Moderate*** Project team cycles back to users to refine requirements, but efforts are taken to limit recursion or revisiting of requirements.	***Heavy***: All development divided into sprints/iteration cycles *"The quarter is divided into three-week sprints." – BigD Project Director*	***Heavy*** Platform functionality evolves through iterative development.
Inherited requirements *(relates to Structural → System-embedded requirements)*	***Heavy***: Project relies upon requirements embedded in the PeopleSoft platform *"The old system had certain structures and the new system has certain structure. If they're going to take full advantage of the new system, they have to redefine things to meet the new structure." – SIS Technical Consultant*	***Limited*** Few inherited requirements with the tools used in development; project focused on a creating a fundamentally new tool	***Heavy***: Requirements 'inherited' from other standards-based products in the Ruby milieu *"A lot of Rubinius work is actually basically reverse-engineering what MRI does, figuring out its behavior." – Rubinius Developer*

While the recognition of specific modes of distribution observed in each of the development environments is important, the more substantive question is how these mechanisms interact to enable a design team to achieve requirements closure. To address this question, we must model the computational structures adopted in each project.

Computational Structures

Our second focus in the analysis was the computational structures observed in the different development environments (RQ2). *Computational structure* refers to the sequence of activities by which information is processed through the propagation of representational state across representational media (Hollan et al. 2000; Hutchins 1995b); that is, the steps that move the cognitive system from raw data to a desirable solution. We model the computational structures of the three cases to support comparison and contrast. In these models, we have attempted to articulate (1) the modes of social, structural, and temporal distribution reflected in each environment and (2) how these distribution mechanisms relate to the essential cognitive objectives of a requirements process – i.e., discovery, specification, and validation & verification of design requirements.

SIS: The computational structure of requirements work in the SIS project reflects a fairly linear process. The first facet of the process focuses on gap analysis, understanding where the features of the selected software platform fail to meet the lived needs of users. The project then moves into a period of cyclical development of specifications for platform modifications. Finally, after specifications are established, the project team again engages with users in a secondary design process in which requirements are validated, verified, and refined. Figure 1 provides a graphical representation of the computational structure.

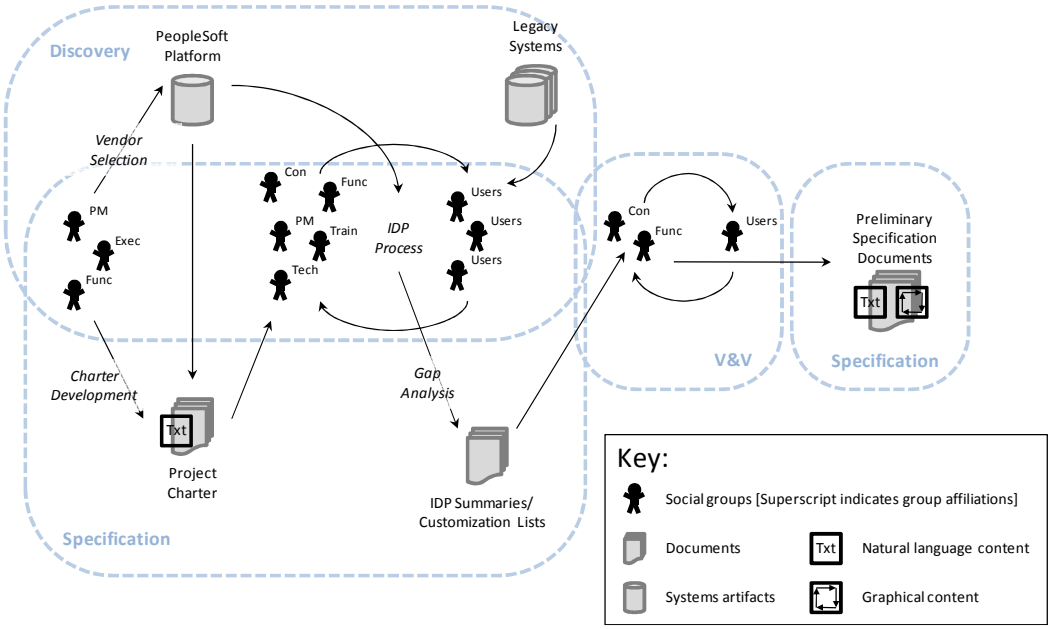

Figure 1. SIS Model: A Structure Development Computational Structure

BigD Project: In contrast to the SIS project, the BigD computational structure is intensely iterative. The cognitive tasks of discovery, specification, and V&V are inherently intertwined in this model. In particular, specification is pursued in lightweight manner with little formalism, but continual use of *ad hoc* artifacts (i.e., mockups and throw-away prototypes) which also support validation of users' preferences. While the process is intensely interactive, the activity is driven by very small teams (two to three individuals) with periodic engagement of other stakeholders. The computational structure is represented in Figure 2.

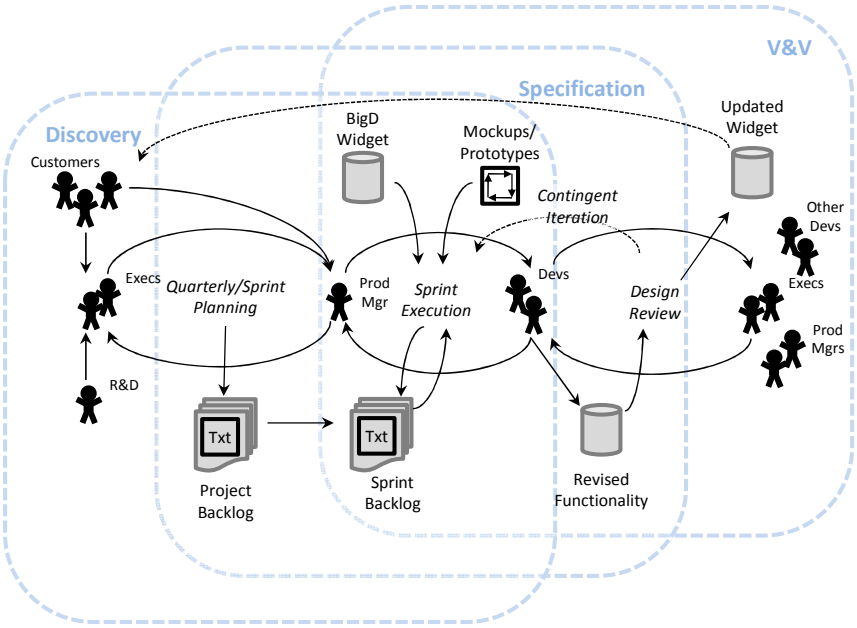

Figure 2. BigD Model: An Agile Development Computational Structure

Rubinius Project: The clearest distinguishing characteristic of Rubinius' computational structure is the central role of structural distribution. All requirements-oriented interaction on the project is executed through system artifacts. Even informal discussions that do not immediately result in changes to the system are coordinated through artificial informalisms, such as IRC chats. While the intense iteration suggests a parallel with the BigD structure, Rubinius does reveal relatively distinct activities oriented toward discovery, specification, and V&V. Figure 3 shows the modeled computational structure of Rubinius.

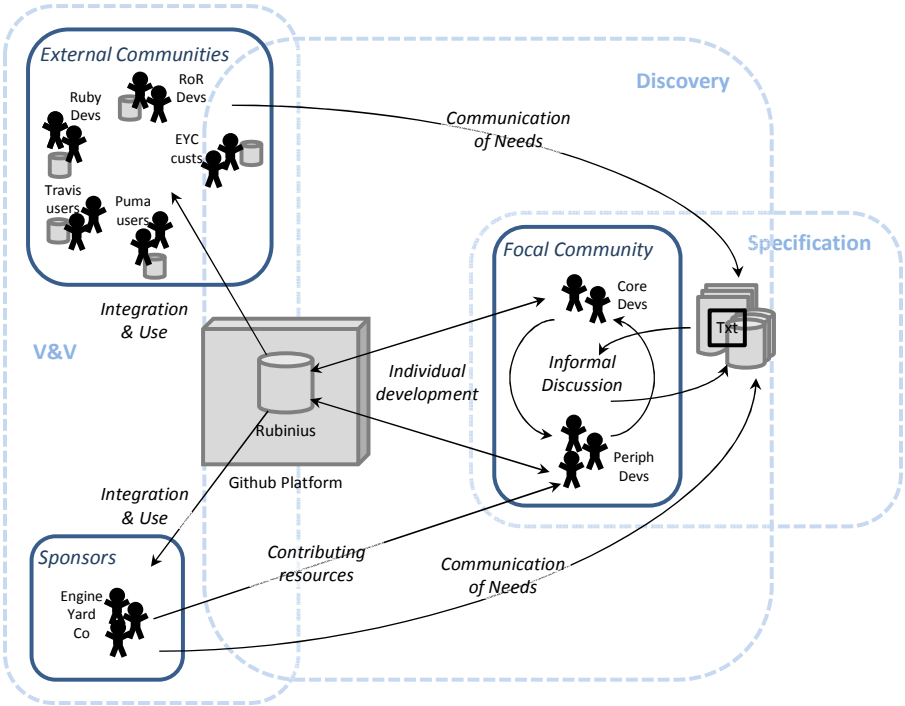

Figure 3. Rubinius Model: An OSSD Computational Structure

Discussion

The modeling of these computational structures enables us to identify important points of comparison and contrast between the three development environments with respect to the dynamic processes of distributed cognition (RQ3). Table 3 summarizes the comparison of the three environments with respect to the distributed cognitive processes outlined by Hansen et al. (2012).

Table 3. Comparison of Computational Structures

Distributed Cognition Dynamics	Projects		
	SIS	***BigD***	***Rubinius***
Knowledge Redundancy (KR)	Collective development of requirements documents establishes KR across team members.	Close collaboration creates KR within sub-teams, but broader KR is only achieved during design reviews.	KR is high because of common skill sets of developers and common access to artifacts.
Shared Understanding (SU)	SU is undermined by strict specialization of roles, but structured walkthroughs help to mitigate this effect.	Emphasis on generalized skill sets and close collaboration supports SU within small project teams.	SU challenged by the individual initiation of changes; may be improved through informal exchange.
Transparency of Action (TA)	Low TA outside of walk-throughs. Team members cannot see actions of others.	High TA achieved through heavy reliance on graphical representations.	High TA, because all work is mediated through commonly-held artifacts.
Cognitive Offloading (CO)	High CO through the reliance on documents and legacy artifacts.	High CO established through prototyping and iterative development.	High CO as work is entirely coordinated through technical infrastructure.

Reviewing this summary, we see some clear points of commonality and contrast. In each of the three methodologies we see high levels of cognitive offloading, with team members using structural artifacts to embody requirements decision-making. Similarly, measures designed to foster knowledge redundancy are observed in each of the projects analyzed. Interestingly, this knowledge redundancy stands in contrast to the traditional observations of thin spread of application knowledge within software development environments. In terms of contrast between the projects, the analysis shows that transparency of action is significantly higher in the agile (BigD) and OSSD (Rubinius) cases than in the structured development environment (SIS). Shared understanding provides another point of contrast, where strict specialization (SIS) and individual initiation of action (Rubinius) presents a challenge to a common understanding, but the emphasis on generalized skills in the agile environment fosters such shared understanding.

While some of the findings regarding differences between the computational structures and distributed cognitive dynamics of the three environments can be inferred given a familiarity with the methodologies espoused, the broader insight is that the modeling of the computational structures provides a novel basis for analysis of these methodologies from a cognitive perspective. Since requirements are a key source of project management challenges, and the distributed cognition framework provides analytical leverage for exploring methodological differences in approaches to this fundamental question, we contend that this theoretically-grounded modeling technique provides a powerful mechanism to advance our understanding of requirements challenges. Importantly, this technique reflects an acknowledgment of the increasingly distributed nature of the phenomenon itself.

Since our findings indicate that the modes and structures of distributed cognition map with varying degrees to attributes of project management methodologies, we believe this framework suggests multiple avenues for future research. First, applying the distributed cognition framework to the existing body of research on project management presents a novel perspective on such persistent requirements research questions as the efficacy of requirements modeling techniques, the role of prototyping in requirements discovery, the promotion of requirements reuse, and scenario-based approaches to requirements engineering. Second, our analysis of the *differences* in modes and structures of requirements cognition implies a theoretically-grounded approach to explaining the *reasons* for the evolution of such methodological structures.

Conclusion

The research in this paper started out with the observations that requirements related activities are increasingly distributed across geographies, organizations, time, and project methodologies. To understand requirements-related challenges, we leveraged a theoretical model based on *distributed cognition*, for use as a novel framework with which to analyze and understand requirements processes. In this framework, requirements processes are viewed as a socio-technical distributed cognitive process, wherein cognitive load is distributed across multiple dimensions of social, structural, and temporal elements. These distribution mechanisms interact to form computational structures that process information of representational states through various workflows, until a requirements closure is achieved.

From a theoretical perspective, our findings underscore the value a distributed cognitive perspective for understanding requirements work within software development projects, even across disparate development environments. Given the variance in computational structures across these environments, we contend that the application of distributed cognition principles provides a fruitful mechanism for evaluating the strengths and weaknesses of diverse software development practices.

References

Aurum, A., and Wohlin, C. 2005. "Requirements Engineering: Setting the Context," in: *Engineering and Managing Software Requirements,* A. Aurum and C. Wohlin (eds.). Berlin, Germany: Springer-Verlag, pp. 1-15.

Boehm, B. 1984. "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software* (1:1), January, pp 75-88.

Borgida, A., Greenspan, S., and Mylopoulos, J. 1985. "Knowledge Representation as the Basis for Requirements Specifications," *IEEE Computer* (18:4), April, pp 82-91.

Boyatzis, R.E. 1998. *Transforming Qualitative Information: Thematic Analysis and Code Development*. Thousand Oaks, CA: Sage Publications, Inc.

Cao, L., and Ramesh, B. 2008. "Agile Requirements Engineering Practices: An Empirical Study," *IEEE Software* (25:1), pp 60-67.

Carroll, J.M. 2002. "Scenarios and Design Cognition," *2013 21st IEEE International Requirements Engineering Conference (RE)*: IEEE Computer Society, pp. 3-3.

Cerpa, N., and Verner, J.M. 2009. "Why Did Your Project Fail?," *Communications of the ACM* (52:12), pp 130-134.

Cross, N. 2001. "Design Cognition: Results from Protocol and Other Empirical Studies of Design Activity," *Design knowing and learning: Cognition in design education* (7), pp 9-103.

Crowston, K., Howison, J., and Annabi, H. 2006. "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice* (11:2), pp 123-148.

Cybulski, J.L., and Reed, K. 2000. "Requirements Classification and Reuse: Crossing Domain Boundaries," *Lecture Notes in Computer Science* (1844), pp 190-210.

Damian, D., Eberlein, A., Shaw, M., and Gaines, B. 2003. "An Exploratory Study of Facilitation in Distributed Requirements Engineering," *Requirements Engineering* (8:1), pp 23-41.

Dorfman, M. 1997. "Software Requirements Engineering," in: *Software Requirements Engineering,* R.H. Thayer and M. Dorfman (eds.). IEEE Computer Society Press, pp. 7-22.

Eisenhardt, K. 1989. "Building Theories from Case Study Research," *Academy of Management Review* (14:4), pp 532-550.

Gacek, C., and Arief, B. 2004. "The Many Meanings of Open Source," *IEEE Software* (21:1), pp 34-40.

Glaser, B.G., and Strauss, A.L. 1967. *Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago, IL: Aldine Publishing Company.

Grünbacher, P., and Braunsberger, P. 2003. "Tool Support for Distributed Requirements Negotiation," *Cooperative methods and tools for distributed software processes*), pp 56-66.

Guinan, P.J., Cooprider, J.G., and Faraj, S. 1998. "Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach," *Information Systems Research* (9:2), June, pp 101-125.

Halverson, C.A. 1994. "Traffic Management in Air Control: Collaborative Management in Real Time," *ACM SIGOIS Bulletin* (15:2), pp 7-11.

Hansen, S., Berente, N., and Lyytinen, K.J. 2009. "Requirements in the 21st Century: Current Practice & Emerging Trends," in: *Design Requirements Engineering: A Ten-Year Perspective,* K.J. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson (eds.). Heidelberg, Germany: Springer-Verlag, pp. 44-87.

Hansen, S.W., Robinson, W.N., and Lyytinen, K.J. 2012. "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering," *Proceedings of the 45th Hawaii International Conference on System Science (HICSS'12)*, Maui, HI: IEEE Computer Society, pp. 5224-5233.

Hickey, A., and Davis, A. 2003. "Elicitation Technique Selection: How Do Experts Do It?," *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*), pp 169-178.

Hollan, J., Hutchins, E., and Kirsh, D. 2000. "Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research," *ACM Transactions on Computer-Human Interaction* (7:2), pp 174-196.

Hutchins, E. 1995a. *Cognition in the Wild*. Cambridge, MA: MIT Press.

Hutchins, E. 1995b. "How a Cockpit Remembers Its Speed," *Cognitive Science* (19), pp 265-288.

Hutchins, E. 2000. "Distributed Cognition," in: *International Encyclopedia of the Social & Behavioral Sciences*. Elsevier, Ltd.

Jarke, M., Bubenko, J., Rolland, C., Sutcliffe, A., and Vassilou, Y. 1993. "Theories Underlying Requirements Engineering: An Overview of Nature at Genesis," *IEEE International Symposium on Requirements Engineering (ISRE'93)*, San Diego, CA: IEEE Computer Society, pp. 19-31.

Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J., and Robinson, W. 2011. "The Brave New World of Design Requirements," *Information Systems* (36:7), pp 992-1008.

Jarke, M., and Pohl, K. 1994. "Requirements Engineering in 2001: (Virtually) Managing a Changing Reality," *Software Engineering Journal* (9:6), pp 257-266.

Kirsh, D. 1999. "Distributed Cognition, Coordination and Environment Design," *Proceedings of the European Cognitive Science Society*, pp. 1-10.

Kotonya, G., and Sommerville, I. 1998. *Requirements Engineering: Processes and Techniques*. New York, NY: John Wiley & Sons.

Larkin, J.H., and Simon, H.A. 1987. "Why a Diagram Is (Sometimes) Worth Ten Thousand Words," *Cognitive Science* (11:1), pp 65-100.

Levina, N., and Vaast, E. 2005. "The Emergence of Boundary Spanning Competence in Practice: Implications for Implementation and Use of Information Systems," *MIS Quarterly* (29:2), June, pp 335-363.

Li, J., Slyngstad, O., Torchiano, M., Morisio, M., and Bunse, C. 2008. "A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components," *IEEE Transactions on Software Engineering* (34:2), pp 271-286.

Loucopoulos, P., and Karakostas, V. 1995. *System Requirements Engineering*. New York, NY: McGraw-Hill, Inc.

Magnus, P. 2007. "Distributed Cognition and the Task of Science," *Social Studies of Science* (37:2), pp 297-310.

Maiden, N. 2008. "Requirements 25 Years On," *Software, IEEE* (25:6), pp 26-28.

Majchrzak, A., Cooper, L., and Neece, O. 2004. "Knowledge Reuse for Innovation," *Management Science* (50:2), pp 174-188.

Mylopoulos, J. 1998. "Information Modeling in the Time of the Revolution," *Information Systems* (23:3-4), pp 127-155.

Piller, F.T., Moeslein, K., and Stotko, C.M. 2004. "Does Mass-Customization Pay? An Economic Approach to Evaluate Customer Integration," *Production Planning and Control* (15:4), pp 435-444.

Rogers, Y. 1993. "Coordinating Computer-Mediated Work," *Computer Supported Cooperative Work* (1:4), December, pp 295-315.

Scacchi, W. 2009. "Understanding Requirements for Open Source Software," in: *Design Requirements Engineering: A Ten-Year Perspective,* K.J. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson (eds.). Heidelberg, Germany: Springer-Verlag, pp. 467-494.

Strauss, A.L., and Corbin, J. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park, CA: Sage.

van Lamsweerde, A. 2000. "Requirements Engineering in the Year 00: A Research Perspective," *Proceedings of the 22nd international conference on Software engineering*), pp 5-19.

Wallace, D.R., and Ippolito, L.M. 1997. "Verifying and Validating Software Requirements Specifications," in: *Software Requirements Engineering,* R.H. Thayer and M. Dorfman (eds.). Los Alamitos, CA: IEEE Computer Society Press, pp. 389-404.

Walz, D.B., Elam, J.J., and Curtis, B. 1993. "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration," *Communications of the ACM* (36:10), October, pp 63-77.

Weick, K.E., and Roberts, K.H. 1993. "Collective Mind in Organizations: Heedful Interrelating on Flight Decks," *Administrative Science Quarterly* (38:3), September, pp 357-381.

Yin, R.K. 2003. *Case Study Research: Design and Methods*. Thousand Oaks, CA: Sage Publications Inc.

Zhu, H., Jin, L., Diaper, D., and Bai, G. 2002. "Software Requirements Validation Via Task Analysis," *Journal of Systems and Software* (61:2), pp 145-169.