Association for Information Systems
AIS Electronic Library (AISeL)

International Research Workshop on IT Project Management 2013

International Research Workshop on IT Project Management (IRWITPM)

12-14-2013

The Structures of Computation: A Distributed Cognitive Analysis of Requirements Evolution in Diverse Software Development Environments

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

Amol Kharabe
Ohio University, kharabe@ohio.edu

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

The Structures of Computation: A Distributed Cognitive Analysis of Requirements Evolution in Diverse Software Development Environments

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

Amol Kharabe
Ohio University
kharabe@ohio.edu

Kalle Lyytinen
Case Western Reserve University
kalle@case.edu

ABSTRACT

Despite decades of research, effective requirements engineering (RE) remains a significant challenge for software development projects. In addition, the study of RE processes has failed to keep pace with dramatic changes in IS development practice. In this study, we frame RE as a fundamentally socio-technical computational task in which diverse social actors and artifacts collaboratively "compute" the requirements for an envisioned software system. Through the perspective of distributed cognition, we analyze the distributed RE activities of three IS development projects, representing distinct methodological approaches. We develop models of the computational structures of these projects to support a novel analytical basis for comparison and contrast of three development methodologies - structured development, agile development, and open source software development.

Keywords

Requirements engineering, distributed cognition, structured development, agile development, open source development.

INTRODUCTION

Requirements engineering (RE) refers to processes by which information systems (IS) designers discover, specify, validate, and manage the functionality that an envisioned system is expected to possess, as well as the constraints to which it must conform (Kotonya and Sommerville 1998). Despite more than three decades of research in the RE domain, requirements-oriented issues remain a primary source of IS project distress (Aurum and Wohlin 2005; Crowston and Kammerer 1998; Hickey and Davis 2003; van Lamsweerde 2000). This persistence of requirements-based challenges has been complicated by the rapidly changing nature of RE *practice* and IS design (Jarke et al. 2011). One of the most notable trends in this regard is that RE activities are increasingly distributed across locations, organizations, participants, and time (Hansen et al. 2009). Such broad distribution poses new challenges and presents a stark contrast to the traditional view of RE, in which a small group specifies a system in relative isolation prior to downstream development and implementation (Jarke and Pohl 1994). In addition, the software development landscape has experienced an explosion of new methods and technologies including the rise of commercial off-the-shelf (COTS) and framework-based development (e.g., SOA), agile development methods, and open source software development (OSSD). Interestingly, some of these new methods (e.g., agile development) have emerged specifically in reaction to the perceived shortcomings of older RE approaches. However, despite these novel approaches, the fundamental challenge of deciding the design needs of an artifact has continued to persist (Cao and Ramesh 2008).

In the wake of these multi-faceted changes, RE research has struggled to keep pace (Hansen et al. 2009; Jarke et al. 2011; Kaindl et al. 2002). While some research has focused on RE in geographically distributed teams (e.g., Damian et al. 2003; Grünbacher and Braunsberger 2003), or comparisons of RE processes across different development paradigms (Gacek and Arief 2004; Scacchi 2009), the extant literature includes relatively little exploration of the diverse ways in which RE is simultaneously distributed across space, organizations, artifacts, and time. While RE 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 of RE parameters affects RE activities and their outcomes (Jarke et al. 2011).

In the current research, we seek to address this gap both *theoretically,* by developing cognitive models of RE computation within distributed projects, and *empirically* by using rich field study data to conduct rigorous analyses of task flows in multiple development environments. Specifically, we adopt a cognitive perspective which focuses on mechanisms by which participants in the RE undertaking process information to arrive at a commonly held understanding of the needs and constraints of stakeholders and users. With reference to Cheng and Atlee's (2009) widely cited roadmap of RE research, we assert that the adoption of a cognitive perspective on RE tasks incorporates two research strategies - *leveraging other disciplines* and *evaluation*. Furthermore, the research is intended to address the "RE research hotspot" (Cheng and Atlee 2009) of addressing emerging needs.

From a cognitive perspective, we frame RE as a fundamentally socio-technical computational task. That is, a development project is a socio-technical cognitive system, composed of diverse social actors and artifacts which collaboratively work to "compute" the requirements for an envisioned 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 may be expected to vary across different development methods (e.g., traditional structured development vs. open source software development). Further, we argue that RE-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 also anticipate that, given the diversity of configurations, computational structures may vary in their effectiveness at achieving such a closure for a given design task. Thus, we seek to address the following key research questions:

- What are the characteristic modes of social, structural, and temporal distribution of RE activities in different development environments?
- How do computational structures vary across different development environments?

To address these questions, we conducted a multi-site field study of software development projects in three distinct development environments – structured development, agile development, and open sourced software development. 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.

BACKGROUND AND THEORETICAL FOUNDATIONS

Requirements Engineering

Whether or not it is explicitly acknowledged by designers, RE represents a fundamental facet of any software development effort wherein several foundational questions are addressed: What do we want to create? What features and functionality should be developed? What practical objectives are we trying to address? Frameworks within the research literature have divided RE into anywhere from two to seven distinct cognitive tasks (Dorfman 1997). For the present discussion, we consider three commonly recognized RE tasks: (1) discovery, (2) specification, and (3) negotiation and prioritization, as they represent distinct and necessary cognitive outcomes for a successful computation of design requirements. RE efforts must address the ways in which software development teams organize, manage, and coordinate these three activities to achieve desired RE 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.

- *Negotiation & Prioritization:* The identification and resolution of conflicts through exploration of the range of design possibilities – generating knowledge of design options and their consequences (Feather et al. 1997; Grünbacher and Seyff 2005; Robinson and Volkov 1998) often through the use of negotiation models (Fisher and Ury 1991). Designers must reconcile requirements due to the presence of disagreements between stakeholders (Grünbacher and Seyff 2005; Robinson et al. 2003), and prioritize the resulting requirements to determine the relative value of individual design requirements (Berander and Andrews 2005; Regnell et al. 2001).

From a cognitive perspective, all software design efforts must address the overall coordination and integration of the three tasks. However, the identification of these tasks reflects no assumptions of their sequencing, executors, means, local goals, or spatial organization. Indeed, as alluded to above, in some development environments (e.g., agile development), designers might demure at the use of the terms we propose vis-à-vis the widely-accepted language of RE. 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 reasoning, are not bounded by the internal mental states of an individual (Hollan et al. 2000; Hutchins 1995a; Hutchins 1995b; Magnus 2007). The development of the theory was motivated by research on teams engaged in complex, multi-person tasks. 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 in these environments 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 in accomplishing 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*).

Social Distribution. Distributed cognition theory contends that cognitive processes are distributed across members of a group. Each individual plays a specific role with respect to the information processing and action of the broader group. This idea has obvious ramifications for the study of distributed RE processes. The vast majority of software design efforts employ a team structure (Guinan et al. 1998). Furthermore, an essential characteristic of development teams is the diversity of knowledge required (Levina and Vaast 2005; Walz et al. 1993).

Structural Distribution. The second implication of distributed cognition is that cognitive processes entail the intertwining of internal and external structure. While traditional cognitive science focuses on the internal states of an individual's mind, a distributed cognitive perspective highlights the ways in which human actors integrate material elements as part of their cognitive processes. The offloading of cognitive demands onto external structure is readily apparent in prevailing RE practice. The heavy emphasis on formal modeling in RE can be understood as a mechanism for creating external structures that support and enhance the human cognitive processes of design.

Temporal Distribution. Finally, distributed cognition theory contends that cognitive processes may be distributed with respect to time. Earlier decisions and actions influence cognitive processes enacted later. Temporal distribution is present in any context where heuristics are used to guide decision making. Design efforts draw heavily upon requirements and artifacts inherited from earlier projects.

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 IS development(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:

The SIS Project. The development efforts which we have dubbed 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 design 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. The consultants included 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 previously worked on the software platform being implemented.

The BigD Project. SocialAgg 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. 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.

The 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, a web-based hosting platform for OSS projects which is based on Git version control system. 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 RE processes. However, the cognitive tasks of RE 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. Relevant areas of focus in the interviews included identification and analysis of processes employed to capture project requirements; the allocation and coordination of RE 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 at each site, additional interview respondents were identified. To ensure multiple perspectives on the software development processes employed on each project, we sought participation from development leads and individual developers. In addition, where applicable, we sought participation of project sponsors or executive stakeholders. All of the interviews were conducted using the aforementioned interview protocol. In all, we interviewed 24 subjects, with the typical duration of each interview 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 RE practices of the projects. The coding was conducted with online, collaborative coding software, called Dedoose. The coding processes addressed two distinct axes of analysis: (1) RE tasks reflected, and (2) modes of distribution. The analysis of RE tasks focused on the processes through which the projects achieved the ends of discovery, specification, negotiation/prioritization, and monitoring. This analysis led 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

The key findings of this research can be categorized into two areas as below – a) cognitive distribution of the requirements along multiple axes, and b) computational structures observed for the distributed cognition of such requirements.

Cognitive Distribution

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. We highlight the distinguishing features of each environment along each of these dimensions.

Social Distribution. The analysis of social distribution mechanisms in the three projects revealed a number of critical dimensions for comparison, including the degree of specialization, the modes of communication between the social actors, and the integration of third-party insights. First, the *degree of specialization* clearly varied across the cases. The SIS project involves significant specialization in both technical and functional areas:

"I think [the Technical Lead's] got that laid out. He's got one developer that tends to do more of the admissions stuff, one in the student financials … So he has segregated duties." – SIS Technical Consultant

In contrast, the BigD project reflects limited specialization in keeping with an agile ideal of broad-based and flexible skill sets for developers. On the BigD project, and indeed throughout SocialAgg's development teams, work is executed in very small collaborative teams:

"My style is to break [the work] down into smaller manageable chunks that can be achieved in the quarter, by no more than two people – one to two people. We call this a one-pizza team, or a two-pizza team, and the basic idea is the team size was just enough to feed them all on either one box of pizza or two boxes of pizza." – BigD Project Director

Finally, because of the voluntary nature of OSSD work, the Rubinius project has specialization on a self-selected basis. However, despite the significant size of the community, our analysis revealed actual specialization to be relatively low in that most members have similar skill sets and professional backgrounds.

In addition to the degree of specialization observed, we see broad variance across the three projects with respect to *communication between stakeholders*. The one-pizza teams of BigD emphasize face-to-face communication reinforced through collocation.

"I think 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

The SIS project instead opts to discuss all requirements changes through a structured walkthrough process. The walkthroughs are conducted by the project's Communications Lead and attended by most project team members. The process is very document-centric, focusing on formal specifications generated by the project's Functional Lead or Consultants (see "Structural Distribution" below).

"I think the walkthroughs were really a good idea. It got people around a table. Some of them were pretty [contentious] … but I think it was what needed to happen." – SIS Communications Lead

In the Rubinius case, almost all communication is computer mediated. In particular, the core and peripheral developers, as well members of external communities using the Rubinius platform (e.g., Ruby and Ruby on Rails developers, Travis CI users, Puma users), communicate with one another through '*informalisms*' such as IRC chats, 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'. It wasn't a direct dare like that but it was basically like, it got me. We were talking about it and it got me thinking and I was like 'Well, I'll just sort of do it as a spike and see how far can I get in the shortest amount of time" – Rubinius Developer

With respect to *third-party sources*, each of the three projects draws requirements knowledge from sources external to the project, but the forms are quite different across the three contexts. In the SIS environment, the heavy reliance upon consultants and the HEUG community provides access to lessons learned at other institutions. The following statement illustrates this dynamic process for integrating externally-generated insights:

"I was encouraging our functional consultants to use [the HEUG community'] and they did … [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. I think it should be used a lot." – SIS Technical Consultant

For BigD, the project benefitted from SocialAgg's explicit focus on the identification of insights or possible avenues for future development from the broader social media and data analytics communities. SocialAgg maintains an engineering unit specifically focused on technology scanning and research and development activities. The role of this group is to call the attention of the firm's executive leadership to promising and innovative developments in the marketplace and to channel such insights to the individual project teams.

"What I mean by 'front end' is not user interface, but it's a leading edge group that is looking at ideas six months, nine months down the line that I could eventual productize, but I need to give them time to think through that stuff and work on it. So it's a small group of three people, but it's a science group and it's applied research. They test ideas." – BigD Project Director

Finally, the Rubinius project draws heavily on insights from communities outside the focal Rubinius community. As noted above, these external communities include other OSS developers and users that focus on distinct platforms that leverage the Rubinius compiler (e.g., Ruby, Ruby on Rails, Travis CI). The Rubinius developers take insights

from this broader community of Ruby-based projects, as well as from developer conferences to identify possible enhancements worth pursuing:

"RubyConf attracts developers, more experienced and knowledgeable developers, so it's a good opportunity to talk to people about a lot of very technical aspects of Ruby and Rubinius, so I do. I talk to a lot of people." – Rubinius Developer

Structural Distribution. Similar to the social distribution, the analysis of structural distribution revealed some relevant bases for comparison, including the role of documentation, the use of graphical representations, and the reliance upon system-embedded requirements. Perhaps the most obvious point of contrast between the three environments centers on their *approaches to requirements documentation*. The SIS project favors formal documentation. All requirements are written up in formal specifications with 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. There's a tracking document that's sort of like the high level tracking document above the spec." – SIS Technical Lead

In keeping with the agile philosophy, the BigD project adopts a lightweight approach, with very limited formal documentation of requirements. The team does create user stories, but these are informal in nature and the team explicitly eschews a document-intensive approach in favor of "just doing what makes sense in a given case."

Similar to BigD, the Rubinius project does not use standardized documentation, the majority of design requirements are embodied in the '*informalisms'* of email and IRC:

"We did coordinate pretty much exclusively on the IRC channel… [T]he actual discussions mostly took place on IRC. I think we had a couple of conference calls maybe, but mostly IRC, and just told people what we were doing, gonna be doing." – Rubinius Developer

With respect to the use of *graphical representations*, we see some similarity between SIS and BigD projects. Both of these projects make extensive use of mock-ups and white board drawings. However, the emphasis on this form of representation as a primary mode of exchange and requirements capture is greater in the BigD context.

"So as the requirements become clearer for some of these larger projects, sprint stories will be entered into FogBugz [the feature tracking tool] against the project, both by the developers and the Project Managers… It's sort of lightweight – it may include mocks, flows, pictures, etc., and so they just get attached to that." – BigD Project Director

In Rubinius, we found little, if any, evidence of graphical representation – the ideas outlined in text-based 'informalisms' get converted right into code by individual developers. This may be due to the fundamental distribution of the team members across both spatial and temporal dimensions, and the inherent limitations imposed by such distributions.

Temporal Distribution. In the area of temporal distribution, we see differences between the three projects with respect to the importance of iteration and the role of "inherited" requirements. The emphasis on iteration is greatest in the BigD project, where the sprint cycles reinforce an evolutionary approach to the development. Each sprint builds upon the affordances created in the previous sprint, while remaining open to changes based on new requirements flowing from the project sponsors:

"The quarter is divided into three-week sprints, so there's typically going to be about four sprints in every quarter. Then in each sprint, the team that is assigned to that sprint of the product line will decide what they want to achieve. So it's a stepwise breakdown from the yearly strategic arc to the quarterly to the three-week sprints." – BigD Project Director

Iteration is similarly critical in the Rubinius project as the platform is incrementally enhanced by the diverse developer community. As is common in OSS environments, the platform is considered to be in a constant state of evolution. This iteration of the design is a fundamental guiding principle.

In the SIS project, iteration is relatively limited. In keeping with a structured development approach, the team is modifying the system for a full-scale roll-out to the university, rather than adopting an evolutionary build process. Interestingly, the project's Technical Lead stated that the limitation of changes to the underlying vendor platform was a primary focus for the development effort:

"I would say I should be trying not *to do much systems development. I sort of judge how well we do by limiting that." – SIS Technical Lead*

Finally, with respect to "*inherited" requirements* (i.e., requirements embedded within the specific systems or tools adopted), in the cases of SIS and Rubinius, the teams are highly reliant upon requirements embedded within the systems they employ. For example, in SIS, a large number of requirements are established *a priori* based on the capabilities of the vendor platform and interfacing needs of the university's existing systems:

"Basically what you find is 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. For example in the legacy system, you can only have three programs of study … and then in PeopleSoft world, that's normalized … You can imagine other things like that where you wouldn't be able to take the data in the old format and put it into the new one." – SIS Technical Consultant

Similarly, in the case of Rubinius, a significant source of 'inherited' requirements was driven by compatibility with existing standards and other standards-based products in the Ruby milieu. In some cases, the project team focused on reverse-engineering from the related systems:

"MRI is the de facto standard. You know there is no language standard other than what MRI does…A lot of Rubinius work is actually basically reverse engineering what MRI does, figuring out its behavior." – Rubinius Developer

In contrast to the SIS and Rubinius projects, the BigD developers have not inherited extensive requirements with the tools used in their development. This difference undoubtedly relates to the fact that the BigD project is focused on a fundamentally new tool while both SIS and Rubinius are modifying an existing platform.

A summary of comparisons of the three projects with respect to their social, structural, and temporal modes of distribution is provided in Table 1.

Table 1. Modes of Cognitive Distribution by Project

Modes of Distribution	Projects		
	SIS	**BigD**	**Rubinius**
Social			
Specialization	High	Low	Limited; self-selection
Communication	Document-mediated; Structured walkthroughs	Face-to-face/co-location	Computer-mediated; Informalisms
Third-Partyies	Consultants; HEUG	Technology scanning	Ruby communities
Structural			
Documentation	High/Formal	Low	Informalisms; Code base
Graphical	Mock-ups; Whiteboards	Mock-ups; Whiteboards	Limited
Temporal			
Iteration	Low	High	High
Inherited requirements	Extensive	Limited	Extensive

Computational structures

In distributed cognition theory, a significant emphasis is placed on the computational structure of a given cognitive system. The *computational structure* is the sequence by which information is processed through the propagation of representational state across representational media (Hollan et al. 2000; Hutchins 1995b) – steps that move the cognitive system from raw data to a desirable solution. Within a given work environment, the computational structure can be either explicit (e.g., a documented standard operating procedure) or implicitly understood by the social actors within the cognitive system (Al-Rawas and Easterbrook 1996).

When considering the distributed cognitive aspects of requirements-oriented processes, we must identify the computational structure of the various systems we analyze. Accordingly, in this section of our findings, we model the computational structures of the three cases to support comparison and contrast. Figure 1 through 3 provide graphical representations of the computational structures for the SIS, BigD, and Rubinius projects, respectively. 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 anRE process –i.e., discovery, specification, and validation & verification (V&V) of design requirements.

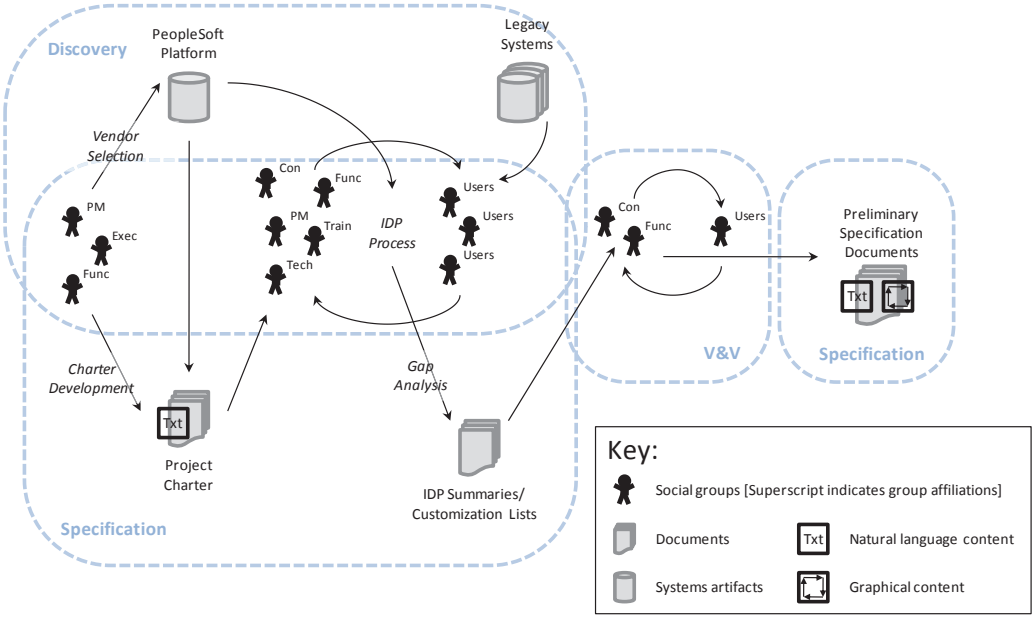

Figure 1. SIS Model: A Structure Development Computational Structure

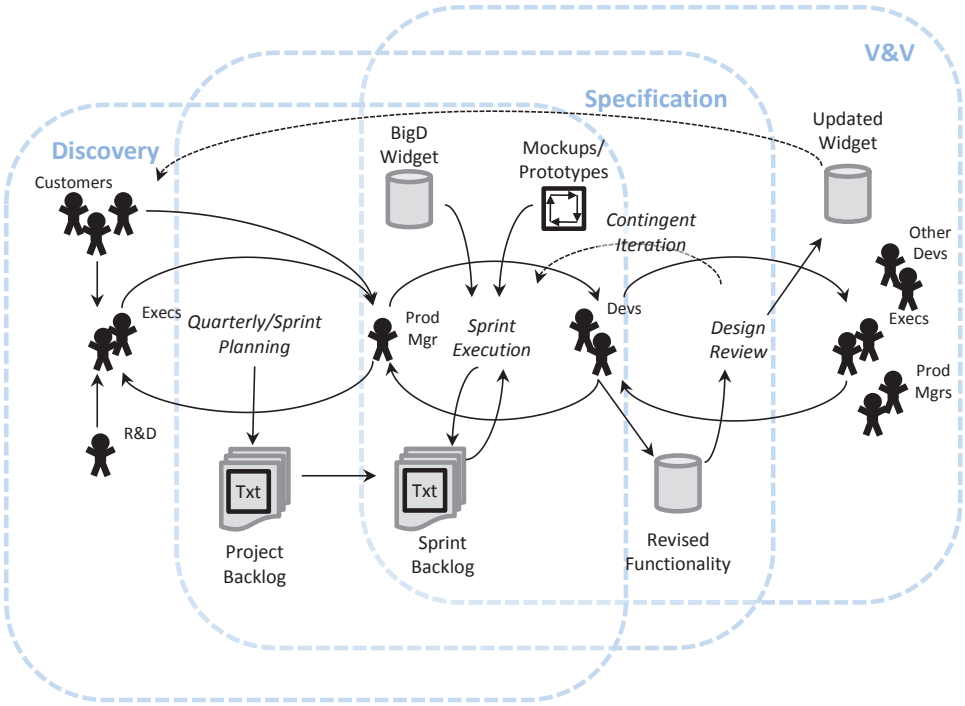

Figure 2. BigD Model: An Agile Development Computational Structure

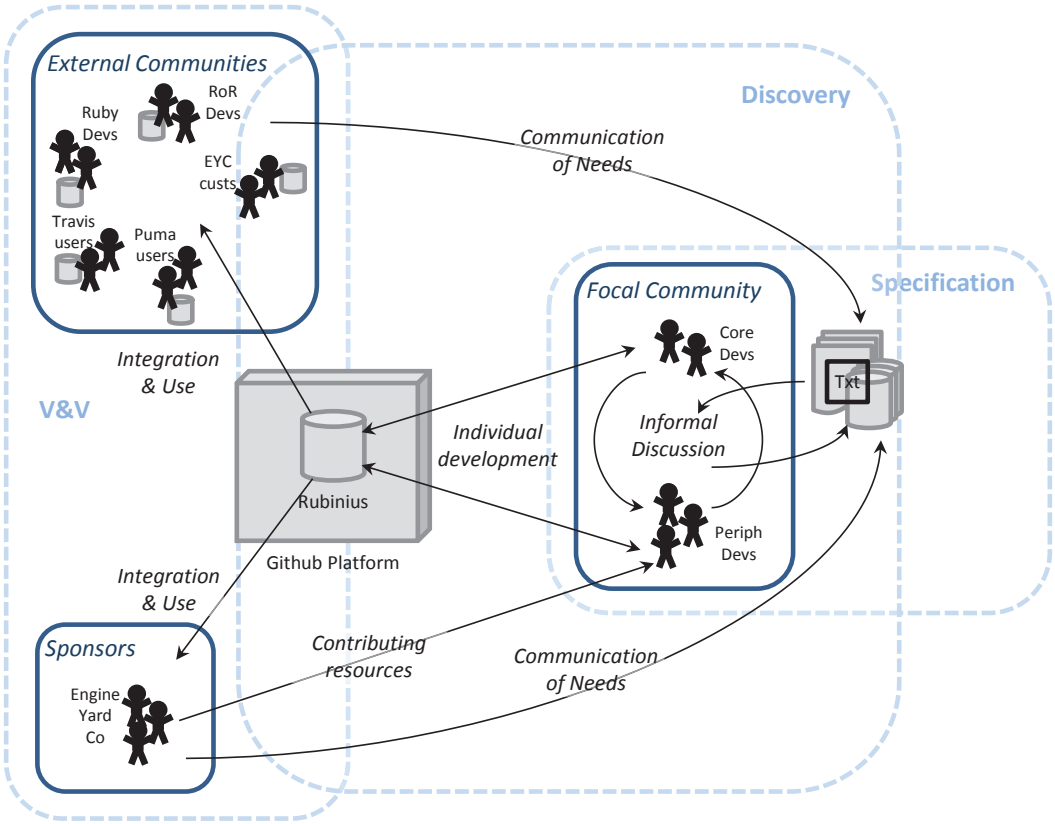

Figure 3. Rubinius Model: An OSSD Computational Structure

Looking beyond the superficial differences between the three computational structures (e.g., idiosyncratic processes and labels), we see some interesting and significant basis of comparison and contrast between the three project environments. First, with respect to comparison, all three of the environments have clear mechanisms through which the objectives of requirements discovery, specification, and V&V are achieved. This is significant since in two of the three environments (i.e., agile→BigD, OSSD→Rubinius) the traditional language of requirements engineering is generally disregarded or devalued. Yet, our analyses illustrate that the objectives traditionally associated with RE (i.e., identifying new requirements, articulating them in a concrete and communicable form, and obtaining feedback on their execution) remain relevant and fundamental to the software development undertaking, even if the more established way of describing those aspects are discarded. Secondly, we see that all three environments involve significant interplay between diverse social actors or stakeholder groups (social distribution) and extensive integration of external representations (structural distribution) to move the system toward effective computation of design requirements. This suggests that the theoretical perspective of distributed cognition provides some useful insight for analyzing diverse software development contexts.

Turning to the points of contrast between the three environments, a number of features are readily apparent. First, the relatively linear nature of the SIS structured development process stands in marked contrast to the other two contexts. While the fact that a structured development effort is more linear than an agile or OSSD process may not be surprising, the computational structure also reveals that this process allows for significantly less feedback and broader community engagement, as the design and development progresses. It similarly reveals that the prominence of gap analysis (i.e., between the acquired platform and specific user requirements) as a primary focus for requirements activity in the structured environment significantly bounds (and furthermore *binds*) the options for system development. Second, we see that agile development environment (i.e., BigD) stands out for the extensive overlap of discovery, specification, and V&V cognitive processes. Nearly every facet of the process involves some elements of these three fundamental objectives. Perhaps tied in with this observation, the obvious emphasis on iteration is readily apparent in the agile computational structure. Finally, while iteration is also prominent in the OSSD context, the most notable feature of this environment is the role of structural distribution. All interactions between social actors in the Rubinius project are mediated by external structure.

While several of the findings regarding differences between the computational structures of these three development environment are not surprising 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 environments and methodologies from a cognitive perspective.

DISCUSSION

The research in this paper started out with the dual observations that a) despite more than three decades of requirements engineering (RE) research, RE project phases are a prime source of issues, gaps and failures in information systems development projects, and b) such RE related activities have become increasingly distributed across geographies, organizations, time and project methodologies. To understand such RE related challenges, we leveraged a theoretical model based on *distributed cognition*, for use as a novel framework with which to analyze and understand RE processes. In this framework, RE is viewed as a socio-technical *distributed cognitive process*, wherein the cognition is distributed across multiple dimensions of social, structural and temporal elements, which interact to form computational structures, that process information of representational states through various workflows, until a final cognitive stage is reached representing requirements closure.

This theoretical model gave rise to our research questions on the applicability of such a model to distinct development environments types, characterized by different project management methodologies, as well as on the potential differences in the computational structures that may evolve to address the characteristics of the varying development environments types. To confirm the validity of distributed cognition framework across multiple types of development, as well as to analyze the resultant computational structure attributes and workflows, we conducted a multi-site field study of three development environments types– structured development, agile development and open-source development.

From a theoretical perspective, our findings validate that distributed cognition is a valid theoretical framework for understanding RE, even across disparate development environments types. Although the degree and extent to which the distribution varied across the social, structural and temporal modes in each of the development environment types, as well as the characteristics and workflows of the computational structures that evolved in each of the

development environment types varied in terms of linearity and overlap across requirements phases, it is quite clear that all three development environment types have a consistent underlying structure which can be theoretically explained by distributed cognition.

Since RE is key source of project management issues and challenges, and the distributed cognition framework can be used to explain and understand RE, it implies that from a practice perspective, distributed cognition is a powerful mechanism to apply to the understanding of project management challenges, especially those that are increasingly being derived from the distributed nature of projects themselves. Specifically, since our findings indicate that the modes and structures of distributed cognition map with varying degrees to attributes of project management methodologies, it implies that project management practice can analyze and apply this framework, to understand the specific conditions and limitations under which the framework can potentially explain the successes and failures of project management.

Both, the theoretical and practical implications above, indicate multiple areas of future research. First, applying the distributed cognition framework to the existing body of research on project management is a potentially rich area of future novel research streams. Second, research can be done to extend the work done in this paper on the *differences* in the modes and structures of cognition, to theoretically explain the *reasons* for the evolution of such modes and structures of distributed cognition. Finally, literature review indicates that prior research in application of distributed cognition framework to RE has been somewhat hampered by the limitations of the ethnographic approach taken in such research. Hence a future area of research, would be to replicate and extend the field study approach of this paper, to further validate such application of distributed cognition framework to RE.

REFERENCES

Al-Rawas, A., and Easterbrook, S. M. (1996) Communication problems in requirements engineering: A field study, in *First Westminster Conference on Professional Awareness in Software Engineering*, Royal Society: London, UK.

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

Berander, P., and Andrews, A. (2005) Requirements Prioritization, in *Engineering and Managing Software Requirements,* A. Aurum and C. Wohlin (eds.), Springer-Verlag, Berlin, Germany, pp. 69-94.

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

Boyatzis, R. E. (1998) *Transforming qualitative information: Thematic analysis and code development*, (Sage Publications, Inc, Thousand Oaks, CA.

Cao, L., and Ramesh, B. (2008) Agile requirements engineering practices: An empirical study, *IEEE Software* (25, 1), pp 60-67.

Cheng, B. H. C., and Atlee, J. M. (2009) Current and Future Research Directions in Requirements Engineering?, in *Design Requirements Engineering: A Ten-Year Perspective,* K. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson (eds.), Springer-Verlag, Berlin, Germany, pp. 11-43.

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.

Crowston, K., and Kammerer, E. E. (1998) Coordination and Collective Mind in Software Requirements Development, *IBM Systems Journal* (37, 2), pp 227-245.

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.

Feather, M. S., Fickas, S., Finkelstein, A., and van Lamsweerde, A. (1997) Requirements and Specification Exemplars, *Automated Software Engineering* (4, 4), pp 419-438.

Fisher, R., and Ury, W. (1991) *Getting to Yes: Negotiating without Giving In*, (Boston, MA.

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*, (Aldine Publishing Company, Chicago, IL.

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

Grünbacher, P., and Seyff, N. (2005) Requirements Negotiation, in *Engineering and Managing Software Requirements,* A. Aurum and C. Wohlin (eds.), Springer-Verlag, Berlin, Germany, pp. 143-162.

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), pp 101-125.

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.), Springer-Verlag, Heidelberg, Germany, pp. 44-87.

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

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. (Year) Theories underlying requirements engineering: An overview of NATURE at Genesis, IEEE International Symposium on Requirements Engineering (ISRE'93), IEEE Computer Society, San Diego, CA, 1993, 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.

Kaindl, H., Brinkkemper, S., Bubenko Jr., J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., Leite, J. C. S. P., Mead, N. R., Mylopoulos, J., and Siddiqi, J. (2002) Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda, *Requirements Engineering* (7, 3), pp 113-123.

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

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), pp 335-363.

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

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.

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.

Regnell , B., Höst, M., Dag, J. N. o., Beremark, P., and Hjelm, T. (2001) An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software, *Requirements Engineering* (6, 1), pp 51-62.

Robinson, W. N., Pawlowski, S., and Volkov, V. (2003) Requirements Interaction Management, *ACM Computing Surveys (CSUR)* (35, 2), pp 132 - 190.

Robinson, W. N., and Volkov, S. (1998) Supporting the Negotiation Life-Cycle, *ACM, Communications of the ACM* (41, 5), pp 95-102.

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.), Springer-Verlag, Heidelberg, Germany, pp. 467-494.

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

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.

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), pp 63-77.
Yin, R. K. (2003) *Case Study Research: Design and Methods*, (Sage Publications Inc, Thousand Oaks, CA.