Abstract

For decades, the identification and management of design requirements have been recurring challenges for software development projects. As development practices have proliferated, research on requirements engineering has struggled to keep pace. In this study, we analyze requirements engineering as a socio-technical computational task in which individuals and artifacts collaboratively “compute” the design requirements for a software solution. Using distributed cognition theory, we analyze the requirements-oriented activities of two successful systems development organizations, each representing a distinct pattern of requirements computation. Our analysis of the computational structures employed in these organizations reveals that they are heavily influenced by their distinct environmental conditions which are associated with divergent requirements knowledge characteristics. Building upon these findings, we develop a cognition-oriented contingency model of RE practices. We conclude with implications for the future theorizing of requirements computation.

Keywords: Distributed cognition, requirements engineering, structured development, agile development, requirements knowledge, contingency theory

Introduction

Requirements engineering (RE) refers to activities by which information systems (IS) designers and developers discover, specify, and manage the functionality that an envisioned software system will possess and the constraints to which it must conform (Jarke et al. 2011; Kotonya and Sommerville 1998). Despite decades of research on RE, challenges arising from the failure to effectively articulate and address desired requirements have continued to plague IS projects (Aurum and Wohlín 2005; Cerpa and Verner 2009). The persistence of requirements-based challenges is further complicated by the rapidly changing nature of IS design (Jarke et al. 2011). For example, requirements-oriented activities have become increasingly distributed across locations, organizations, stakeholder groups, and time (Hansen et al. 2009), posing new challenges and presenting a significant contrast to the traditional view of RE where a small group of analysts specifies a greenfield 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, 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. However, despite the emergence of new approaches, the fundamental challenge of determining whether or not a piece of software meets the users' needs persists (Cao and Ramesh 2008).

In the wake of these changes, RE research has struggled to stay abreast of practice (Hansen et al. 2009; Jarke et al. 2011). While some research has focused on RE in geographically distributed teams (e.g., Damian et al. 2003; Grünbacher and Braunsberger 2003) or across development paradigms (Gacek and Arief 2004; Scacchi 2009), RE research has provided relatively little exploration of the ways in which requirements activities are organized across space, organizations, artifacts, and time based on the nature of the requirements knowledge that needs to be captured in the final system. By requirements knowledge, we refer to the “the implicit or explicit information that is created or needed while engineering, managing, implementing, or using requirements, and that is useful for answering requirements-related questions in any phase of a software project” (Maalej and Thurimella 2013: p. 7). As this definition suggests, requirements knowledge is a broader concept than the mere requirements themselves. It also incorporates properties of the user, organizational, and technical environments, such as the volatility of a set of requirements as influenced by the dynamics of an organizational context (Eriksson and Goldkuhl 2013). While past RE research maintains a strong design science focus on prescribed tools and techniques, the field has been marked by relatively limited theorizing that connects specific RE practices to the associated base of requirements knowledge (Jarke et al. 1993). Consequently, we know little about how a change in a given property of requirements knowledge (e.g., the volume of the requirements sources) affects the organization of RE efforts and their outcomes (Jarke et al. 2011).

In this research, we seek to address this gap both empirically and theoretically. Empirically, we use two rich case study data sets to conduct rigorous analyses of variations in RE under different contingencies (i.e., requirements knowledge properties). Theoretically, we develop a cognitive framing of RE based on the principles of distributed cognition and show how the configurations of socio-technical systems pursuing stable requirements under different contingencies are organized radically differently. Specifically, we adopt a cognitive perspective which focuses on diversified mechanisms by which participants in RE process information so as to arrive at an understanding of the needs and constraints of their development projects.

We posit that RE is central to both problem formulation and solution generation during “design cognition” (Carroll 2002; Cross 2001). We also contend that RE encompasses fundamentally cognitive processes that are widely distributed, heterogeneous, and emergent. From a cognitive perspective, we approach requirements analysis as a socio-technical computational task. That is, a development project forms a socio-technical cognitive system which is composed of heterogeneous social actors and artifacts which interact to “compute” the requirements for a desired state of a software system (Hansen et al. 2012). We use the term requirements computation to mean the process of moving from an ambiguous problem state to a solution state in which requirements are viewed as clear, agreed upon, and implementable. The concept is employed in this research to reinforce the cognitive framing of RE efforts as a mechanism of socio-technical information processing. In this view, requirements are determined using various computational structures – i.e., processes for exchange between elements of a cognitive system – and they are expected to vary based on different properties of the underlying requirements knowledge (e.g., large vs. small volume, high vs. low volatility). Finally, we contend that, in light of the diversity of configurations, computational structures will vary in their effectiveness at achieving requirements computation. Thus, we seek to address the following key research questions:

- What are the characteristics of requirements knowledge that influence requirements computation?
- What are the characteristic modes of social, structural, and temporal distribution of RE efforts in different environments?
- How do these computational structures differ across dimensions of requirements knowledge?

To address these questions, we report a multi-site field study of software design organizations in two distinct development environments – structured development and agile development. In analyzing these environments, we draw upon the Theory of Distributed Cognition (Hollan et al. 2000; Hutchins 1995a) to identify the characteristic modes by which requirement computation is achieved.
Requirements Engineering

Whether or not explicitly acknowledged by designers, requirements represent an essential facet of any software development project. That is, any design effort must address the following fundamental questions at some point: What do we want to create? What features and functionality should the system possess? What objectives guide the design effort? Depending upon how these questions are addressed the research literature has divided RE activities into multiple tasks (Dorfman 1997), typically centering on three critical facets: (1) discovery, (2) specification, and (3) validation and verification:

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

Specification: The rendering of discovered requirements into some representational form, so that the knowledge can be validated for correctness, clarity, feasibility, and coherence, and thereafter used in downstream design (Borgida et al. 1985; Mylopoulos 1998; van Lamsweerde 2000b). Specification is the point of transition where articulated needs are extended with explicit design implications.

Validation & Verification (V&V): The activities used to assess the degree to which RE has been conducted effectively and supports downstream development. These two terms are generally grouped together. Validation reflects the process of ensuring that requirements accurately reflect the intentions of the stakeholders including their prioritization by negotiation (Zhu et al. 2002), while verification focuses on the degree to which requirements conform to established standards of requirements quality (Boehm 1984; Wallace and Ippolito 1997). Boehm (1984) captures the distinction succinctly with his questions: “Am I building the right product?” (validation) and “Am I building the product right?” (verification).

These three tasks are understood to represent distinct and necessary elements of a successful requirements computation. RE efforts thereby must address the ways in which software teams can manage and coordinate these three activities to achieve desired outcomes. However, the identification of these tasks reflects no assumptions of their temporal execution/sequencing, actors involved, means, local goals, or spatial organization. Indeed, 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. We will next review the features of requirements knowledge which may give rise to different forms of discovery, specification, and V&V.

Characterizing Requirement Knowledge: Six “V”s Model

In their exploration of the ‘New RE,’ Jarke and Lyytinen (2015) argue that the field “has shifted from managing internal complexity to adapting and leveraging upon external and dynamic complexity” (p. 11:1). Consequently, many of the established concepts around requirements quality demand a new look. Classically, RE has focused on consistency, correctness, and completeness of requirements documents, placing the focus on specification and validation tasks (Davis 1993; Zowghi and Paryani 2003). From the perspective of the ‘New RE,’ addressing multiple properties of complex requirements knowledge is central: “Whereas most of the interest in the past focused on understanding and managing the inner and static complexity of the design task by using abstraction, modularization, and related principles, today’s complexity is of a different ilk. It is also external and dynamic” (p. 11:2). To characterize this new environment, they propose six dimensions of requirements knowledge (conveniently, all starting with the letter “V”) on which requirements qualities have changed in the ‘New RE’ context. Table 1 presents a summary of the Six-V model from Jarke and Lyytinen (2015).

Some RE researchers may take issue with the characterization of the “Old RE view” of some of the ‘V’s, such as vagueness and variance, which are supported by existing research and tools (Burns and Stalker 1961; Jarke et al. 2011; Jarke and Lyytinen 2015; Woodward 1965). Most, however, would agree with the general view presented: RE needs are likely to be different if the dimensions of the environment change as they give rise to very different types of cognitive tasks. We will next review how to characterize these cognitive tasks in understanding RE as a process of distributed cognition.
<table>
<thead>
<tr>
<th>Dimension</th>
<th>Definition</th>
<th>Old complex Requirements</th>
<th>New complex requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume</td>
<td>The size of the requirements pool influencing the scope of the work</td>
<td>Major focus of RE as influences effort estimation Medium to Large</td>
<td>Significant during RE as influences effort estimation (Hansen et al. 2009); Large to Ultra-large</td>
</tr>
<tr>
<td>Veracity</td>
<td>To what extent requirements express the needs of the stakeholders and are consistent</td>
<td>Emphasized as the key feature of RE task, works well if requirements can be frozen</td>
<td>Important as an ideal but not key feature of most RE efforts (Jarke et al. 2011; Ross 1977)</td>
</tr>
<tr>
<td>Volatility</td>
<td>The rate at which the requirements change over a given period of time</td>
<td>Recognized as a key reason for the failure of waterfall, e.g. (Brooks 1995)</td>
<td>Constant feature of software development for most environments (Hansen et al. 2009)</td>
</tr>
<tr>
<td>Vagueness</td>
<td>To what extent designers and other stakeholders understand the content and consequences of the requirement</td>
<td>Not recognized as an important element other than to be avoided during RE task</td>
<td>Inherent feature of many RE initiatives due to initial lack of user learning or understanding of the dynamism introduced by the software in the environment</td>
</tr>
<tr>
<td>Variance</td>
<td>The variation in the design scope and consequences of the requirement pool and the heterogeneity of design components involved</td>
<td>Not recognized as an important element in RE activity</td>
<td>Significant element influencing RE dynamics and complexity (Hansen et al. 2009; Jarke et al. 2011)</td>
</tr>
<tr>
<td>Velocity</td>
<td>The rate at which requirements are changing over time</td>
<td>Not important and recognized</td>
<td>Significant contributor at specific context of RE especially in software platforms (Fisher et al. 2013)</td>
</tr>
</tbody>
</table>

Table 1. Six-V Model of Requirements Knowledge (adapted from: Jarke & Lyttinen, 2015)

**Distributed Cognition**

*Distributed cognition* refers to a branch of cognitive science, which 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 such settings, information processing is not limited to individual actors; rather, it is distributed across members of a group and elements of their operating environments, with a significant portion of the cognitive load being borne by the artifacts used by group members. In cognitive science, *cognitive load* refers to the amount of information processing effort required of a cognitive system to perform a particular task (Hollan et al. 2000; Paas et al. 1994). This can also be understood as the total computational effort needed to move from a problem state to a solution state.

In a seminal articulation of distributed cognition theory, Hutchins (1995a) frames cognition as “the propagation of representational state across representational media” (p. 118). This framing extends the unit of analysis for cognitive activity from the individual to the entire group accomplishing a given task. With this fundamental shift in perspective, the theory makes three central assertions about cognition (Hutchins 2000): 1) cognitive processes are distributed among the 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 2 provides a summary of these mechanisms and their applicability in software development environments.
In addition to the basic structures of cognitive distribution, Hansen et al. (2012) call attention to dynamic processes that are characteristic of distributed cognitive systems. These are defined through heuristics which are defined as general rules for sequencing and pacing computational activity and to guide its direction. Distributed cognitive systems typically manifest redundancy of knowledge, shared understanding to cognitive transformations, transparency of action, and varying degrees of cognitive offloading. Knowledge redundancy refers to collaborative systems having a degree of overlap in the knowledge of actors, enabling the failure of any single element to be compensated for by the knowledge and experiences of other elements (Hutchins 1995a). Shared understanding means that effective collaborators must be aware of how each contributes to achieving the broader system’s 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 can be transferred from human actors to artifacts through structural distribution (Larkin and Simon 1987). That is, cognitive offloading occurs when artifacts or elements of the natural environment (i.e., external structural elements) are used to transform cognitively-intensive tasks (e.g., remembering a complex itinerary) into comparatively simple ones (e.g., responding to an audible meeting reminder). By simplifying the information processing of social actors, the external structural elements bear a portion of the cognitive load for the broader system.

### Research Design

#### Multi-Site Field Study

In this research, we conducted a multi-site field study of ongoing systems development organizations. This multi-case approach enables us to engage in a rich exploration and theorizing of the socio-technical, cognitive process of practicing information systems development (ISD) professionals (Eisenhardt 1989; Yin 2003). Specifically, our analysis focused on two organizations employing two distinct development approaches: structured, platform-based development and agile software development. The site inquiries were conducted in accordance with prevailing case study canons, including theoretical purposeful sampling, 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, members of the research team collaboratively developed a case study protocol, which included the research questions guiding the inquiry, the essential design of the study (i.e., multiple case analyses 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, technologies or artifacts used by organizational members, and perceived challenges to effective RE. 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.

Data Collection and Analysis

On initial engagement with each of the focal organizations, we performed an in-depth, semi-structured interview with a key respondent with oversight authority for the systems design and development functions of the organization. Key areas of focus in the interviews included identification and analysis of processes employed to capture 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, additional interview respondents were identified. To ensure multiple perspectives on the software development processes employed, we sought participation from development leads, individual developers, business and systems analysts, and product managers. Where applicable, we also sought participation of project sponsors or executive stakeholders. All of the interviews were conducted using the aforementioned interview protocol. In the broader research program with which this study is associated, over 50 interviews were conducted. For the focal organizations in this study, we conducted interviews with four representatives of each organization for a total of eight interviews for the present analysis. The average duration of the interviews was approximately 80 minutes, with individual interviews ranging from 70 to 110 minutes. To enable triangulation across multiple sources of evidence, our interviews were augmented with the collection and analysis of artifacts that supported the interaction of organizational members (e.g., project documentation, requirements specifications, and communication media).

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 (Boyatizs 1998; Braun and Clarke 2006). While the thematic analysis was conducted in line with several 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 multiple a priori frameworks, including the Six-V model of requirements knowledge (Jarke and Lyytinen 2015) and the theory of distributed cognition (Hollan et al. 2000; Hutchins 1995a). In an initial open coding phase, all data were coded and parsed into categories to identify important themes around the RE practices of the organizations. The analysis of RE tasks focused on the processes through which the organizations achieved the ends of discovery, specification, and V&V in their ISD projects. In axial coding, the analysis focused on the determination of relationships between themes identified in open coding. In addition, modes of cognitive distribution and requirements knowledge properties of the respective environments were assessed in the axial coding phase. Finally, in the selective coding phase, the dimensional categories established in the axial coding phase were refined and consolidated to identify the central theoretical elements discussed in our findings. As the result of the three analytical phases, we determined forms of distribution within and across tasks as they evolved through discovery, specification, and V&V in confronting distinct operating environments. All coding was conducted using NVivo, a qualitative data analysis software.
Case Analyses

SocialShare

SocialShare is a Silicon Valley-based social media aggregation firm that develops software for both internal and external clients. The firm’s core product is an all-in-one customizable widget, which enables Internet users to share content with friends, family, and associates via email, instant messaging, and text messages. The primary external customers for SocialShare’s services are online content producers (referred to as “publishers”), who integrate the SocialShare widget on their sites to enable sharing by users. The SocialShare widget is currently integrated on more than two million publisher sites and over one hundred social media channels. In addition to its widget, SocialShare offers a wide range of social data analytics services to its publisher customer base. The firm provides publishers with a customizable dashboard, which highlights a wide array of analytical content, including social referrals and sharing destinations, traffic from various social media channels, and an index of sharing quality. With the growth of its social analytics offerings, SocialShare has become a significant player in big data analytics.

The focus for software development activities at SocialShare is determined through planning activities over multiple timeframes. At the highest level, the firm establishes its areas of strategic focus on an annual basis. Based on the strategic focal points established for the year, the firm determines their development goals for each product or service line on a quarterly basis. Managers and development teams do not plan detailed goals more than one quarter out. Armed with the quarterly goals for each product or service line, project teams are formed to develop concrete development objectives for the quarter. At this point in the planning process, the SocialShare teams engage in an agile iteration planning process.

SocialShare’s development teams employ an agile development approach that is a variation on the Scrum methodology (Schwaber 1995). The firm is very committed to the agile philosophy, but it is not dogmatic in their application of the Scrum technique. 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. At the end of each three-week sprint, the firm conducts design reviews. During the design reviews, each development team will present to the broader organization (i.e., other development teams as well as senior managers) on the work they have completed and the planned activities for the next sprint. The design reviews are considered an important mechanism for keeping organizational members informed and “on the same page” with respect to development efforts.

RE practices within SocialShare center on regular engagement with customers and close interaction between development team members. The primary point of contact with the customers for the various types of product and service offerings is the role of the Product Manager. Product Managers consistently interact with customers and communicate information about desired functionality to the development team. However, other members of the development group are expected to join the Product Managers in customer conversations on a regular basis. This practice is explicitly focused on ensuring that team members have a deep understanding of customer needs and changing business environments. In addition, the firm leverages multiple social media channels to receive feedback from customers around their perceptions of SocialShare’s products and services. The most prominent such channels include Twitter, Facebook, and LinkedIn feeds. All SocialShare employees are encouraged to monitor these channels and respond to inquiries where appropriate. Finally, communication of requirements knowledge within the teams is fostered by maintaining extremely small project teams. All projects are undertaken by “one-pizza or two-pizza teams” (i.e., groups small enough to be fed with one or two pizzas in a sitting). In practice, these teams consist of one to three developers working in collaboration with a Product Manager. The rationale behind this small team structure is that it enables consistent communication and collaboration around team tasks. By avoiding large/distributed teams, SocialShare promotes rapid cycles in identifying and addressing customer requirements and challenges.

---

1 In order to respect the assurance of confidentiality to study participants, pseudonyms are employed.
FieldServe Technologies

FieldServe Technologies is a leading provider of field service software for mobile workers in a range of industries across the globe, with a particular emphasis on the telecommunications sector. The software is delivered in a cloud-based software-as-a-service (SaaS) model. The FieldServe software enables corporate clients to automate the scheduling, routing, dispatching, and outcome monitoring functions for their mobile field service personnel. In addition, the software provides forecasting, service optimization, and job allocation analysis to support continuous improvement of field service effectiveness.

Software development activities within FieldServe center on necessary modifications to the firm’s established field service platform, EnactFSP to make it fit the client’s needs. Most development work is conducted on a project-by-project basis with individual clients. The structure of FieldServe’s development group reflects strict role segmentation, with key roles including project manager, solutions architect, sales engineer, business analyst, development engineer, and quality assurance (QA) specialist. In the case of the development engineering and QA functions, units are globally distributed with bases in the United States, South America, and Eastern Europe. In line with the firm’s clear role specialization, FieldServe development follows a firm-specific execution of a structured development (i.e., waterfall) approach. Desired modifications are communicated to the development teams and all functionality is developed and approved by QA before being integrated in the final implementation solution for a given client.

RE practices follow well-defined and documented procedures in the FieldServe environment. The initial requirements discovery is executed by a sales engineer prior to the finalization of a contract with a potential client. This process consists of the sales engineer documenting anticipated modifications based on the specific needs of the client firm. Once a contract is finalized, FieldServe sends the client firm a Customer Questionnaire, which is used to capture key aspects of their current-state business processes in advance of large group setting. Following the receipt of the questionnaire, a detailed discovery and specification process is enacted. FieldServe personnel (i.e., project manager, sales engineer, solutions architect, and business analyst) meet with representatives from the client firm for an intensive requirements workshop that generally runs for two to three days. This workshop essentially centers on a gap analysis, with the EnactFSP platform acting as a baseline solution to develop a specification of the final system. The workshop also identifies areas where the existing platform fails to meet the unique needs of the client and defines them as future requirements. Throughout this process, the default assumption is that the EnactFSP platform embodies the vast majority of the client's requirements and that the system will only be modified, if deemed truly necessary and a viable process-oriented solution cannot be identified, and the client is willing to cover the cost of developing those components.

The requirements workshop results in a formal specification document, the Business Requirement Specification (BRS). The BRS is typically 100 pages in length and includes detailed descriptions of the client’s current business processes, the proposed architecture for the solution, and use case models. Finally, the FieldServe technical personnel use the BRS to create a Software Requirement Specification (SRS), which captures the detailed technical requirements for the proposed system. This document is then transmitted to the development teams for downstream development. As this summary highlights, FieldServe RE processes entail clearly defined discovery, specification, and V&V phases, with a heavy orientation towards standardized documents.

Case Comparison

A consideration of the SocialShare and FieldServe cases provides a stark contrast with respect to RE practices and broader software development approaches. It is important to note that both of these firms have been extremely successful in their respective domains. Thus, the contrast between RE approaches in these cases is not a comparison of more or less advantageous methods, but rather that of similarly effective methods in markedly different environments. Accordingly, we first highlight the distinct requirements knowledge characteristics encountered by the firms. We then explore the ways in which the organizations differently distribute cognitive load of computing requirements in their respective contexts.
Requirements knowledge properties

The analysis of the cases reveals significant differences in the properties of requirements knowledge addressed by the two firms. Operating in the dynamic domain of social media and big data analysis, SocialShare is faced with requirements knowledge that is ill-structured and rapidly changing. Indeed, the SocialShare respondents openly conceded that the precise nature of the requirements is not well understood at the outset of development iterations. In terms of the Six-V model (Jarke and Lyytinen 2015), SocialShare confronts requirements with high degrees of vagueness and volatility. With respect to vagueness, the firm expends significant effort at the beginning of each quarter clarifying ambiguous requirements:

“The purpose of that first sprint is to know what the requirement is ... We don’t know fully what’s the requirement, but we just know the product goal. So the purpose of that is discovery.”
– CTO

Regarding volatility, the SocialShare respondents frequently remarked upon the changing nature of the requirements within the market that they serve. This volatility is given as a primary reason that the organization did not focus on long-term requirements determination:

“Our approach is more of just-in-time, rather than requirements months and months ahead of time, because the market changes, so you have to redo them.” – Product Manager

By contrast, FieldServe operates in an environment where the majority of requirements need to be well-understood and crystallized from the inception of a project. At the same time significant attention must be paid to gaining a shared understanding between FieldServe personnel and those of their client firms on what these requirements are. In the Six-V model, the vagueness of the FieldServe marketplace is quite low as the firm knows the requirements its system can meet. In contrast, the focus for FieldServe personnel is on the veracity and volume of the requirements knowledge. The question of veracity is reflected in the firm’s heavy emphasis on achieving alignment of understanding through large group discussions and thorough documentation:

“At the end of the requirements-gathering workshops, we get to a first version of the BRS and privately we take one more week to write down all the use cases into the BRS. After that we take one more or two weeks to review the BRS internally with the Technology Center and when we all agree that the BRS is okay, we go to the client again on site and review the BRS with the client and get the sign-off. So it never takes less than six or seven weeks.” – Delivery Manager

In addition to veracity, FieldServe confronts a significant volume of requirements knowledge. In moving from initial scoping to a final SRS document, a large number of explicit requirements are captured for each project:

“Typically say one to three weeks on-site with the customer, depending on the complexity and the scope of the modules they have purchased, and we issue what’s called a ‘Customer Questionnaire,’ which can be tailored based on the modules they bought, that we ask the customer to fill in. This questionnaire basically makes them document the key aspects of their business so that we don’t have to spend time on-site asking about things that can be asked off-line.” – Senior Project Manager

Using the relative frequency of coded segments for each of the Six-V properties, Figure 1 provides a graphical representation of the requirements knowledge characteristics encountered by the two firms.
This model accentuates the fundamental differences in the nature of requirements knowledge encountered in the two settings. The question then becomes how the firms structure their cognitive systems to address these differential properties?

**Distribution of cognitive load**

As we noted, the effective determination of design requirements is a cognitive process executed by a distributed cognitive system of actors, artifacts, and the social and cognitive routines they employ. Given the differential nature of the requirements confronted by the two firms, we would expect that the ways in which they distribute the cognitive load would similarly differ. Table 3 provides a summary of distinct forms of social, structural, and temporal distribution in the two cases.

<table>
<thead>
<tr>
<th>SocialDistribution</th>
<th>SocialShare</th>
<th>Illustrative Statements</th>
<th>FieldServe</th>
<th>Illustrative Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Social Distribution</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limited role specialization; individuals expected to work across roles</td>
<td>&quot;It's critical to have also all the members of the team interact with the customer, because they get to see firsthand what are those problems, what are their needs.&quot; – Product Manager</td>
<td>Significant role specialization; clear demarcation of duties</td>
<td>&quot;I think it's the Solution Architect [who is] the star in the requirements-gathering phase. In the planning phase, the star is the Project Manager. In the configuration phase, it's the Business Analyst.&quot; – Project Manager</td>
<td></td>
</tr>
<tr>
<td>Small, collocated teams with close collaboration</td>
<td>&quot;Typically, my style is to break work down into smaller manageable chunks that can be achieved in the quarter, by no more than two people - one to two people.&quot; – CTO</td>
<td>Large group workshops with representation from all parties</td>
<td>&quot;Depending on the topic, you may have a smaller workshop or a bigger workshop, but oftentimes it is the broad group, 'cause then everybody get to understand the hand-to-hand solution.&quot; – Senior Project Manager</td>
<td></td>
</tr>
<tr>
<td>Organizational design reviews at the end of each sprint</td>
<td>&quot;One of the things that’s important is to keep the organization informed, so at the end of each sprint, we will have a full update by every team where they talk about what they achieved in that sprint.&quot; – Engineering Manager</td>
<td>Requirements verification at the end of each phase</td>
<td>&quot;We have checkpoints where we meet with the client. So for the beginning of the workshops in our requirements-gathering sessions, for the review of the BRS, for integration testing, for the UATs at the goal line, we travel to the client site and work together.&quot; – Project Manager</td>
<td></td>
</tr>
</tbody>
</table>
### Structural Distribution

<table>
<thead>
<tr>
<th>Lightweight documentation; low formality</th>
<th>“I think we are not that document-heavy ... We do whatever makes sense in the presentation of the design.” – Engineering Manager</th>
<th>Extensive documentation; gap analysis centers on formal documents</th>
<th>“The BRS document [is] typically about a 100-page document that contains a description of the customer's current business and the future business in terms of future architecture, as well as the use cases.” – Senior Project Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation of prototypes for communicating design conceptions</td>
<td>“So they end up working extremely closely together, but they are focused on what the architecture and design would be. They may even be doing prototyping, very fast prototyping.” – CTO</td>
<td>Requirements discussions focus on use-case models</td>
<td>“They are described in the form of use cases. It's like use case by use case, it describes how people will operate after our system is implemented, and this includes all the steps and interaction between people and computer systems.” – Chief Technology Officer</td>
</tr>
<tr>
<td>All development tasks captured and tracked in the feature tracking system.</td>
<td>“Sprint stories are entered into FogBugz, both by the developers and the project managers. So they are working on the same system and for every story it actually has its own compendium.” – CTO</td>
<td>RE tasks managed through an open-source ticketing system</td>
<td>“It's called Mantis. I think it’s an open-source, web-based tool. So Mantis is used to capture questions once we are deep enough into the reviews, and you simply create a ticket.” – Senior Project Manager</td>
</tr>
</tbody>
</table>

### Temporal Distribution / Heuristics

<table>
<thead>
<tr>
<th>Iteration and progressive elaboration of design solutions</th>
<th>“So our requirements are becoming clearer. The sprint task gets defined and added to the backlog as we learn more about it in that first sprint.” – CTO</th>
<th>Prominent reliance on requirements embedded within existing systems</th>
<th>“There are certain use cases that almost never change. The process is so standard that it remains the same for the different customers.” – Senior Project Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded requirements in vendor platforms</td>
<td>[On a vendor platform for customer feedback:] “In general this problem is becoming more and more common ... So when we've discovered this platform, it made a lot of sense to just use that.” – Engineering Manager</td>
<td>Emphasis on requirements reuse</td>
<td>“So we have our use case templates. We have use cases from previous projects, and we have standard use cases. So whenever we go in to a customer and start the detailed requirements-gathering, we can come with templates for the standard processes.” – Senior Project Manager</td>
</tr>
<tr>
<td>All development to be executed in three-week sprints</td>
<td>“We don’t do two-week sprints. I think that’s the general practice for many other companies. We like three weeks because that’s what my team is most comfortable with.” – CTO</td>
<td>Tightly organized processes for requirements gathering based on past projects with a span of several months</td>
<td>“Two to four weeks prior to the kickoff, we have identified our team. We have asked the customer to identify their Project Team, and we organize a couple of mobilization goals where we’ll explain our methodology. We’ll explain kind of what’s next … and how we conduct or requirements-gathering.” – Senior Project Manager</td>
</tr>
</tbody>
</table>

### Table 3. Modes of Cognitive Distribution by Case

With respect to social distribution of cognitive load, we see a sharp contrast between SocialShare’s small team structure with limited role specialization and FieldServe’s large group planning with strict role specialization. In responding to the vagueness and rapidly changing nature of their requirements, SocialShare employs their “one-pizza team” idea to engender open exploration of design requirements and fast cycling through possible solutions. In the FieldServe context, the need for clarity of understanding across roles with respect to a large number of specific requirements engenders formal processes for communication in tightly-bounded exchanges.
In the area of structural distribution, SocialShare’s processes again favor informality – using “whatever makes sense” to communicate ideas. Prototypes are used to capture design concepts and create a focus for the small team discussions. Documentation is explicitly devalued, because the vagueness and volatility of requirements implies that documents are likely to lose validity rapidly as understandings (and the associated requirements) change. The opposite is true in FieldServe, where explicit documentation is central to the requirements processes. Low vagueness means that requirements are likely to remain stable as they are documented, and the formal structure creates a mechanism for dealing with the requirements volume and addressing questions of requirements veracity during sign-offs.

Finally, the temporal distribution in the two environments is again markedly different. SocialShare’s processes and decision heuristics emphasize rapid iteration and progressive elaboration of requirements. This reflects a recognition of the fact that the development team cannot be expected to understand the design requirements at the outset (vagueness); rather, they must progress toward requirements clarity through multiple cycles of design, trial-and-error searches, and discussion. In FieldServe, we see a significant focus on reuse of artifacts (e.g., use-cases) and processes derived from past projects. This emphasis mitigates the challenge posed by requirements volume as it enables project members to focus primarily on those requirements that are new or idiosyncratic to the current project. They also serve an important memory function, reducing the likelihood of losing any specific requirement.

The configuration of social, structural, and temporal forms of cognitive distribution in these cases reveals fundamentally different computational structures and related temporal patterns – how processes for exchange between elements of a cognitive system ebb and flow over time (Hansen and Lyytinen 2014). FieldServe embraces formality in all three dimensions, and places a particular premium on structural distribution of requirements knowledge in the form of extensive documentation and a sequential flow of reconfiguring the system in phase-like fashion. In contrast, SocialShare eschews formality in favor of close interpersonal exchange and intense iteration where the patterns are similarly recursive forms of iteration. The past success of both firms implies that the effectiveness of these computational structures reflects a relatively good fit with the requirements knowledge characteristics and their stark differences.

**Discussion**

Our analysis reveals a sharp contrast in the requirements knowledge characteristics confronted in the two cases, as well as corresponding differences in the computational structures mobilized to process the cognitive load along social, structural, and temporal dimensions. In the SocialShare environment, high levels of vagueness and volatility of requirements knowledge are confronted. In contrast, FieldServe encounters low levels of vagueness and volatility, but high levels of concern around the volume and veracity of requirements knowledge. Despite these differences, both cases reveal a conceptual matching between the requirements knowledge characteristics addressed and the computational structures employed. This suggests that a contingency perspective might help us to better understand the relative benefits of different computational structures in distinct design environments.

Classical contingency theory has its roots in explaining the impact of external circumstances on patterns of effective management within an organization (Burns and Stalker 1961; Lawrence and Lorsch 1967; Woodward 1965). It has since been extended in both the software development and information systems disciplines. In the software development space, contingency theory has been used to analyze both internal integration between developers and external integration with users (Alter 1978; McFarlan 1981), as well as to provide guidelines on RE processes as a means to reduce task uncertainty (Davis 1982; Fazlollahi and Tanniru 1991). In the broader information systems domain, contingency theory has been applied to questions of risk analysis and mitigation practices in information systems development contexts (Lyytinen 1988; Ropponen and Lyytinen 2000).

Building upon the work of Hickey and Davis (2004), who suggested the use of contextual factors in understanding RE processes, we propose that contingency theory provides a theoretical prism for explaining the fit between requirements knowledge properties and the adaption of computational structures for processing requirements, for both SocialShare and FieldServe. Specifically, based on our observations in these cases, the management of specific facets of requirements knowledge seems to underlie the design of processes and the behaviors of actors in the cognitive framework of requirements
processing. Accordingly, we propose that a requirements knowledge-based model of contingency theory can help explain the path to requirements computation in both cases.

In a related model of RE, Mathiassen et al. (2007) propose a contingency model for the management of requirements risks. Specifically, they articulate three categories of such risks: identity risks, volatility risks, and complexity risks. **Identity risks** are high when the requirements are “unknown or indistinguishable” (Mathiassen et al. 2007). **Volatility risks** refer to the stability of requirements, with high risks associated with dynamically changing requirements. Finally, **complexity risks** refer to the “understandability of requirements” (Mathiassen et al. 2007). Across all three categories, higher risks correspond to greater difficulty and time consumption in understanding and specifying the requirements for a system. Applying this perspective to the present analysis, we see clear parallels with each of these forms of risk, with identity risk corresponding to **vagueness**, volatility risk having a natural parallel in **volatility**, and complexity risk matching the Six-V concept of **veracity**. A summary of requirements knowledge properties/risk profiles is provided in Table 4:

<table>
<thead>
<tr>
<th>Case</th>
<th>Requirements Knowledge Properties</th>
<th>Prominent Modes of Cognitive Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vagueness</td>
<td>Volatility</td>
</tr>
<tr>
<td></td>
<td>Identity Risk</td>
<td>Volatility Risk</td>
</tr>
<tr>
<td><strong>SocialShare</strong></td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>FieldServe</strong></td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

| Social: Small collocated teams; close collaboration; limited role specialization |
| Structural: Rapid prototyping; lightweight documentation |
| Temporal: Iteration and progressive elaboration of design solutions |
| Social: Large group workshops; strict specialization of roles |
| Structural: Extensive formal documentation; emphasis on use-case models |
| Temporal: Phase-based segmentation of work; requirements re-use |

**Table 4. Contingency Analysis of the Cases**

Thus, the current analysis reveals significant parallels with the contingency model of RE offered by Mathiassen et al. (2007). However, the computational structures observed in our cases also suggest opportunities for the extension of the earlier work.

First, the cases highlight the value of adopting a cognitive approach to the exploration of requirements knowledge. In their contingency theory, Mathiassen et al. (2007) focus primarily on a matching of requirements risk types with established RE techniques. Our analysis reveals that the relevant contingency factors in such an analysis may extend much further than RE techniques to include the mechanisms for interpersonal exchange within the design environment (e.g., large-scale collective gatherings vs. one-on-one collaboration, generalized vs. specialized skill sets; i.e., social distribution), the selection and use of artifacts (e.g., document-intensive vs. prototype-oriented approaches; text-based vs. graphical representations; i.e., structural distribution), and the temporal sequencing and heuristics guiding the design effort (e.g., phase-based vs. iterative sequencing; explicit rules vs. simple heuristics; i.e., temporal distribution). While established RE techniques may relate to each of these dimensions in various ways, the disentanglement of the modes of cognitive distribution may provide an additional degree to analytical leverage. Specifically, this perspective can support a better understanding of how characteristics of the design environment (e.g., tasks, routines, physical layouts) relate to the cognitive challenges confronted within the relevant organizational context.

Second, the cases suggest the potential for contingency considerations **within** the computational structures of an organization. In the two cases, we see an alignment between the modes of cognitive
distribution. For example, in the SocialShare setting, the consistent preference for a lightweight, “whatever-makes-sense” approach to documentation (structural distribution) is directly tied to the small team structure where communication is informal, discursive, and rapid (social distribution). In turn, both of those elements are aligned with the iteration at the heart of the firm’s temporal sequencing of work (temporal distribution). We see similar alignment in the FieldServe case, where the specialization of work, heavy documentation, and phase-based approach to RE are dynamically interrelated. These findings suggest that the successful management of requirements knowledge necessitates a fit between elements across modes of cognitive distribution. The model that we propose provides a means of identifying critical interrelationships and assessing the degree of fit that exists within a given setting.

Finally, the application of the Six-V model (Jarke and Lyytinen 2015) suggests additional properties of requirements knowledge that are not currently addressed in the contingency model developed by Mathiassen et al. (2007). While our case analyses do not support speculation on all of the elements in the Six-V model, the property of volume offers a valuable example. The volume of requirements knowledge was revealed to be a critical factor in the FieldServe environment, guiding much of the firm’s RE practice. It is not clear how this factor would be subject to analysis using the risk categories of identity, volatility, and complexity. A similar consideration might be raised for the remaining facets of the Six-V model. In particular, the identification of additional settings which vary substantially on the axes of velocity and variance could be extremely valuable.

Thus, by integrating a cognitive perspective on RE with a more nuanced delineation of the properties of requirements knowledge, the present research proposes a valuable extension to the contributions of Mathiassen et al. (2007). Of course, a number of limitations must be acknowledged. While we employed selective sampling to identify successful organizations employing distinct ISD approaches, the incorporation of both additional development environments and less successful organizations might enrich the contingency perspective advanced. In particular, a full application of the Six-V model will require the identification of settings where issues of variance and velocity of requirements knowledge are prominent. In addition, the incorporation of more varied development methodologies could help to disentangle the effects of individual requirements knowledge properties (e.g., differential impacts of volatility and vagueness or volume and veracity). Finally, the consideration of “less successful” organizations (i.e., those that have struggled with the management of requirements knowledge) could be invaluable in assessing issues of alignment between modes of cognitive distribution or the interplay of requirements knowledge properties. Indeed, each of these axes is being pursued in the broader program of the current research team.

**Conclusion**

The research reported in this paper is motivated by the observation that RE practices are increasingly distributed across geographies, organizations, time, and project methodologies. To explore the factors affecting management of requirements knowledge, we blend a cognitive perspective on the phenomenon of RE with a nuanced understanding of the requirements knowledge properties in contemporary software design. By understanding RE practice as a socio-technical distributed cognitive process, wherein cognitive load is distributed across social, structural, and temporal elements, we pursue a break with an idealized and outdated view of RE. Through a multiple case study analysis, we identify the ways in which cognitive distribution mechanisms vary across RE contexts, forming distinct computational structures that process and manage representational states related to requirements through various activities until a stable requirements set is achieved. From a theoretical perspective, our findings underscore the value of a (distributed) cognitive perspective to understand requirements work within software projects that operate in disparate environments. Given the observed variance of such computational structures across environments, we contend that the application of a contingency-based distributed cognitive model provides a fruitful avenue to evaluate the strengths and weaknesses of varied RE practices.
References


