Traveling of Requirements in Development of Packaged Software: The Role of Uncertainty and Work Design

Research-in-Progress

Thomas A. Gregory
Georgia State University
Atlanta, GA 30303
tom@alt-tag.com

Lars Mathiassen
Georgia State University
Atlanta, GA 30303
lmathiassen@ceprin.org

Abstract

Software requirements are created, shared and translated across software organizations, and express task uncertainties that software developers need to address through appropriate structuring of processes and the surrounding organization. Using the theory of work design we propose an in-depth qualitative inquiry into development of packaged software for the utility industry. Using the particular context of software provider GridCo, we examine how requirements are constructed, shared, and translated as they travel across vertical and horizontal boundaries. In addressing these questions, we seek contribution to theory by uncovering knowledge about the sources of and responses to task uncertainty in development of packaged software. We also contribute to practice by providing accounts of an organization’s contextual responses to managing requirements as they travel across boundaries, and reaffirm the need for process reinforcement to support the role of boundary spanners. Preliminary indications suggest hardware dependencies and process ill-adapted to software projects contribute to uncertainties.

Keywords: Case study, Software development, Organizational design
**Introduction**

Software is inherently complex (Brooks 1987), making its development a highly risky (Boehm 1991) and uncertain (Mathiassen and Pedersen 2008) activity. Yet, the outcomes of software development, as with development of any other product (Henderson and Clark 1990), are affected in multiple ways by the organizational context in which it is developed. We set out with the assumption that software requirements, and how they are constructed, shared and translated across the software organization, are expressions of task uncertainty faced by software developers. These uncertainties need to be addressed through appropriate structuring of the process and the organization at large. Requirements are necessarily interpreted, negotiated, and translated as they move through an organization on a journey intended to resolve the gap of uncertainty between customer needs and market options, one the one hand, and released software on the other. Thus, we approach the management and organization of software development as a complex human activity from the perspective of software requirements, using as lenses of contingency theory and work design.

Sinha and Van de Ven (2005) argued for reopening the study of work design within and between organizations, and provided a brief review of contingency theory. Contingency theory suggests that organizations build structures and processes to adapt to tasks and contexts (Drazin and Van de Ven 1985), and the consequences of these structures and processes are expressed as tradeoffs between mutually desirable but exclusive goals. Thus, products developed in and between organizations, including software, may reflect attributes of the processes used to create them. More particularly, as requirements travel across organizational boundaries, the selection, negotiation and interpretation of requirements is likely to change depending on how the development activity is structured. Using a specific corporate context of recurrent development of packaged software, this study addresses the problem organizations face in ensuring requirements are effectively created, shared, and translated as they travel across vertical and horizontal boundaries.

In order to respond to this general theme, it is necessary to first examine how requirements behave in a particular organizational context:

**RQ 1:** How are requirements created, shared, and translated in development of packaged software?

Zooming in (Nicolini 2009) on the relationship between work design and requirements, we adopt the “traveling” metaphor (Czarniawska and Joerges 1996; Nielsen et al. 2013) and the notion of boundaries (Carlile 2002) to examine how requirements change within and across boundaries in their journey towards software delivery. This leads to the second research question:

**RQ 2:** How do requirements travel across vertical and horizontal boundaries in development of packaged software?

In addressing these questions, we seek contribution to theory by uncovering new knowledge about the sources of and responses to requirement uncertainty in development of packaged software. In this way, we answer recent calls (Austin and Devin 2009) for inductive qualitative research of design of software processes based on contextual factors. Further, we seek contribution to practice by providing accounts of an organization’s contextual responses to managing requirements as they travel across boundaries, reaffirming the need for process reinforcement to support the role of boundary spanners.

**Background**

Uncertainty is no stranger to software development (Brooks 1987; Zmud 1980), and has many facets. Broadly speaking, uncertainty is the absence of complete information, and has been called the primary issue facing senior managers (Nidumolu 1995; Thompson 1967). Researchers have adopted numerous categorizations and types of uncertainty. Mathiassen, et al. (2007) separated uncertainty into *identity*, the knowing of requirements caused by communications gaps with customers; *volatility*, the changing of requirements whether for internal or external reasons such as time and budget or changes in market and customer preferences; and *complexity*, which reflects difficulty in specifying and communicating requirements, as well as the cognitive load required to understand the effects of implementation. In
framing uncertainty as an information processing problem, Galbraith (1973) suggests the management of uncertainty is one of the purposes of organizations, which can be responded to with differentiation or integration strategies. More recently, Sinha and Van de Ven (2005) rephrased these dimensions as vertical (decomposition) and horizontal (modularity) work design problems as they argued for reopening the study of work design in and between organizational units.

Galbraith (1973, p. 5), defines a particular flavor of uncertainty, called task uncertainty, as “[T]he difference between the amount of information required to perform the task and the amount of information already possessed by the organization.” Software development is the act of creating information (as represented by, for example, source code), and is, by its nature, uncertain, and so a simpler but perhaps less precise definition of task uncertainty in a software context is the difference between what has been done and what has yet to be done in response to specific customer needs and market demands. This difference is expressed in agreed upon requirements and the status of the next software release. Task uncertainty is subtly different from requirements uncertainty, as found in software development literature, meaning “the difference in the information necessary to identify user requirements and the amount of information possessed by the developers” (Nidumolu 1995, p. 136). Although this definition is useful, it too narrowly considers only the relationships between users, requirements, and developers. Task uncertainty, an earlier and more general concept, captures the essence of requirements uncertainty while encompassing the ambiguities certain to arise as ideas travel across an organization.

Galbraith (1973) argues for a correlation between the amount of information to be processed and the level of task uncertainty. By this definition, the potential for uncertainty for new software projects grows with the size of the software being modified, as the amount of information needing to be processed—for example, the size of the code base—increases over time, and the possibility of unintended interactions between modules increases. The same argument applies to an increased number of strategic customers whose demands must be satisfied: as their number increases, the potential for unintended negative interactions increases.

Conversely, the potential for uncertainty decreases as the amount of information already possessed by the organization increases, for example, knowledge as represented by the code base as well as the institutional knowledge of developers over time reduces overall uncertainty. Although over time, uncertainty is expected to go both up and down simultaneously, the resulting dynamics suggests a net increase in uncertainty over time. Task uncertainty is also a function of the way the organization is structured; the design of an organization itself leads to additional uncertainties (Sinha and Van de Ven 2005), and thus as the organization grows uncertainty will tend to increase (as evidenced by coordination cost, e.g., Kraut and Streeter 1995), although this can be mitigated with strategies such as vertical information systems. However, any mitigation of uncertainty is limited by the cognitive ability of participants (Galbraith 1973).

**Packaged Software**

The study of packaged software (Carmel and Becker 1995) is a distinct subset of IS development literature. Packaged software is one of many terms with similar meaning in literature. It is usually contrasted with custom development. Packaged software imposes different demands and constraints on the development process that are not found in all settings, such as time-to-market pressures, particularly at the industry and firm level (Sawyer 2000). Further, packaged software is subject to incremental releases as a result of recurrent development. Within the requirements engineering literature, the notion that packaged software behaves differently is being accepted (Regnell et al. 2001), who also suggests recurrent development has a complicated relationship with time pressures. As (Xu and Brinkkemper 2007, p. 533) point out, “The boundaries distinguishing shrink-wrapped software, commercial off-the-shelf software (COTS), packaged and commercial software are blurred, but the principle of ‘Make one, sell many’ is a common to them all.”

In the software engineering and requirements literature packaged software is sometimes referred to as market-driven software (e.g., Karlsson et al. 2007) or as COTS (Commercial Off-The-Shelf). No standard empirical definition of COTS exists, although Torchiano and Morisio (2004) adopted a broad definition for their empirical study (“[software] acquired from a vendor and used as-is or with minor modifications” p.90), and COTS software has been described as systems which meet the following criteria (Basili and Boehm 2001): a) the vendor controls development, b) the software serves multiple customers (non-trivial
install base), and c) the buyer has no access to developed source code, although this aspect has been challenged (Xu and Brinkkemper 2007). In addition, it has been hypothesized that COTS products typically have a new release every eight or nine months, although there is wide variation (Basili and Boehm 2001).

The vast bulk of COTS research is regarding development of systems with COTS components, rather than of the COTS software itself, so its applicability to this research is limited. It does, however, provide validation for the claim that development packaged software imposes unique contextual constraints as compared with software development generally, or “one-and-done” internal software projects commonly reported on in IS literature.

Although packaged software has demonstrably unique characteristics, it is still considered “a poorly understood phenomena in the information systems research community” (Light and Sawyer 2007, p. 527). A special issue of the European Journal of Information Systems (EJIS) in 2007 brought attention to the issue, but packaged software remains poorly represented in published research.

Requirements

Requirements development techniques in IS have been classified as discovery, prioritization, experimentation, and specification techniques (Mathiassen et al. 2007), which align closely with similar categories in software engineering literature (e.g., SWEBOK 2013). This paper is less concerned with discovery that often occurs via connections beyond firm boundaries (as discussed in the robust requirements elicitation literature), but focuses on those interactions that exist within the firm’s processes, such as requirement exposition and travelling, and translation of requirements into software. We do not naïvely dismiss external market or customer interactions as unimportant, but in scoped this research to be within the firm take the view that uncertainties will emerge as multiple stakeholders interact during requirements construction, and that requirements construction can be sufficiently represented by internal personnel that engage with stakeholders external to the organization. Despite this narrower view, this focus maintains alignment with the accepted requirements engineering dimensions of specification, representation and agreement (Pohl 1994).

The requirements literature is robust, as evidenced by specialized journals in the software engineering discipline. However, despite the desire of much SE literature to treat requirements as infallible directives, IS researchers know that requirements have inherent uncertainties, and reflect the culture, knowledge, and (possibly flawed) interpretations of those writing them (King 2013). Many requirements models show the steps of elicitation, analysis, specification, and validation as discrete and sequential, when they typically occur iteratively and in parallel (Hickey 2004).

Within the software development literature, there is discussion about the most appropriate organizational structure for software organizations (Austin and Devin 2009). While often framed as a tension between plan-based and flexible software processes (Harris et al. 2009), software processes are adopted based on organization strategy and goals (Slaughter et al. 2006), which reinforces the notion that these discussions of the development process are, at their core, work design issues and attempts to mitigate the uncertainty inherent in software development. Much of the IS research on requirements uncertainty adopts Galbraith’s (1973) information processing approach, and demonstrate a negative relationship between requirements uncertainty and project performance (Jiang et al. 2009).

The contingency theory of organization structure was a response to addressing uncertainty in organizations. Although interest in contingency theory waned in the late 70s, there have been several recent calls in top journals (Sinha and Van de Ven 2005; Zammuto et al. 2007) to reapply contingency theory to modern, technology-enabled organizations. The core of contingency theory is that task uncertainty, as originally described by Galbraith (1973) and others, leads to contingent organization structures. Requirements perhaps best reflect formal identifications of task uncertainty in a software development organization, as well as being points where uncertainty is revealed.

Requirements align to the definition of task uncertainty, in that they express a gap between the current state and the desired outcome. We therefore adopt the view that requirements—as documented and stored in an information system—are expressions of task uncertainty. In doing so, we examine the theoretical consequences of task uncertainty theories expressed by contingent organization design.
Framing

Utilizing Galbraith (1973) and Mintzberg (1980) as a basis for describing an organization’s structure, combined with the more recent language of boundary objects (Carlile 2002) and vertical and horizontal (and network) work design (Sinha and Van de Ven 2005) provides a rich description of an organization’s sub-units, boundaries and processes in specific language that reveals theoretically predicted consequences which can be compared with case results and compared against contextual factors to theorize structural causes and potential solutions for requirements uncertainty.

Organization Structure

In describing organization design strategies, Galbraith (1973) lists several alternatives, most of which can be characterized as facets of the horizontal-vertical labels used by Sinha and Van de Ven (2005). The first three strategies relate to vertical structures, and comprise a “mechanistic bureaucracy.” First, “rules or programs” are imposed on sub-units as a standardized way of coordinating their work. Rules fill the same roles for organizations as habits do for individuals, and are particularly useful for repeated work (Galbraith 1973, p. 10). Such rules or programs, however, require attention and reinforcement by hierarchical authorities tasked with reinforcing processes (Mintzberg 1980). The resulting assumption is that procedures are directed rather than organically coordinated between units. The hierarchy (Galbraith’s second strategy), responds to new situations not covered by rule or tradition, and is expected to respond in a way that considers all affected sub-tasks. Thus, hierarchy, which is the epitome of vertical work design, is used to coordinate “in addition to, not instead of, the use of rules” (Galbraith 1973, p. 12). Targeting or goal setting is a third method employed by vertical coordinators in work design, whereby outcome boundaries (such as goals, requirements, schedules, and design constraints) are set as boundaries for the task, and the organizational unit need not seek approval for work within those boundaries. Additionally, Galbraith (1973) lists four response strategies intended to address failings in the mechanistic model. Creating slack resources, through reducing the level of performance required of an organizational unit, and creating self-contained units (that cross functional boundaries) are two strategies designed to reduce the need for information processing and coordination between units. Similarly, the strategies of investment in vertical information systems and the creation of lateral relations are intended to increase the information processing capacity of units. (Galbraith 1973) The four response strategies are intended to be an exhaustive description of an organization’s possible responses to uncertainty, with slack resources (reduced performance) occurring by default.

In their call for reopening the study of work design, Sinha and Van de Ven (2005) not only build on the work of early contingency theorists such as Galbraith (1973) and Mintzberg (1980), they also highlight three types of categorical issues relevant to organizational researchers. In particular, they view division of work as hierarchical, modular, or network. Modular division, also called horizontal division of work is a modular approach to imposing boundaries on tasks; such boundaries are often related to knowledge or function, and are intended to break work into independent pieces. Hierarchical division, or vertical division of work, denotes subordination and an authority relationship, and encompasses the hierarchical structure of coordination and approval. At times, these dimensions interact to form complex network problems (Sinha and Van de Ven 2005). In each of these cases, a knowledge boundary exists between the divisions of work that must be crossed for successful coordination.

Boundaries

Requirements engineering recognizes the dimensions of process, specification, representation and agreement (Pohl 1994). However, in any organizational structure, requirements must cross organizational boundaries, whether hierarchical or functional. The dimensions presented by Carlile (2002) are useful tools in examining the boundary-spanning journey of requirements. In introducing boundary objects, Star and Griesemer (1989) described the interaction of organizations with different agendas, cultures, and viewpoints. These organizational boundaries arise as consequences of horizontal or vertical work design (Sinha and Van de Ven 2005).
The “Travelling” Metaphor

In explaining the travel of ideas metaphor, Czarniawska and Joerges (1996) argue that in order to be useful, management ideas are sent to places other than where they emerged. Along the way, these ideas are translated into new kind of objects, and this translation is a necessary step in their travel. Czarniawska (2009) summarizes how ideas are changed as they move from place to place, arguing the sharing of an idea requires it be newly interpreted. Interpretation and reinterpretation occurs every time an idea moves from one place to another or from one point in time to another. Even when captured in an information system it is still (re)interpreted as the idea passes from the system to the user. In each time or place, the idea is recreated differently. Although this concept of interpretation is broadly described in the traveling literature and organization studies as “translation,” in practice, particularly in the context of requirements and software development, expressing the traveling of ideas simply in terms of translation (in its original meaning) is overbroad. To compensate, this research adapts the traveling framework using concepts from the knowledge management and requirements literature.

Carlile (2002) described how knowledge may have syntactic or semantic aspects. Syntactic boundaries may be represented by source code, formal specification languages, domain specific languages, or more generally, a common lexicon shared by a group. To share an idea is to transfer it across a social boundary while preserving the lexical context used to express it; sharing occurs with a common syntax. However, even with a common syntax, semantic differences arise (Carlile 2002); sharing of knowledge may lead to differing interpretations between the sender and the receiver, for as Czarniawska (2009) acknowledges, “a thing moved from one place to another cannot emerge unchanged: to set something in a new place or another point in time is to construct it anew.” (p. 425) An idea may be translated from one syntax to another. This is a much narrower definition of translation than applied by Czarniawska (2009). When a developer expresses a requirement in code, the idea expressed in the requirement is newly expressed using the syntax of a programming language. Just as with sharing, translation also results in change to the idea. Translation may uncover ambiguous meanings; just as linguistic translation can introduce or mask connotations, so too can syntactic translation of ideas. As already noted, requirements, expressions of ideas, undergo change —unintended or not—as they are specified. Specification and other uncertainty reduction activities construct, or flesh out an idea. This occurs through investigation and elicitation (Hickey 2004), and is a consequence of work done within or across boundaries. Thus, these concepts of constructing, sharing, and translating describe the activities of ideas as they travel through organizations.

Research Design

Method

This research uses a single-site qualitative case study, which is useful for studying contextual factors, particularly organizational structure (Yin 2009). The blend of technical and human-behavioral aspects of software development lends itself to qualitative study (Seaman 1999).

Setting

The research occurred at a medium-sized development arm of GridCo, a large multi-national provider of power and smart-grid solutions. The company produces a product ecosystem of utility meters, network storage and routing components, and command-and-control software that must operate not only on legacy systems, but interoperate with competitors systems and adhere to common standards using a variety of communications mediums (e.g., Internet, radio frequency, power-line carrier and cellular). The development arm is composed of several hundreds of people, and builds hardware, firmware, and administrative control software via a hybrid process of plan-based and flexible development, using more than 35 small software development teams at multiple locations in the U.S. and an offshore captive, as well as outsourced development providers.

For several reasons, the industry is dominated by a handful of incumbents (including GridCo). Because customers tend to be large utility providers, the potential market is limited, and there is thus strong competition for a relatively small number of customers. Because of the customer-centric approach, the command-and-control system developed at GridCo is consistent with Sawyer’s (2000) description of packaged software.
This research is primarily concerned with a single 26 week release cycle of command-and-control system during 2013, but meetings from related projects are being observed as part of data collection, in order to build a richer picture of the release cycle or to observe behaviors similar to those employed but which could not be directly observed for timing reasons. The considered command-and-control system release has multiple dependencies in related hardware and firmware projects, which are increasing the volatility and complexity of the observed requirements.

**Data**

Data collection is occurring over an eight month period, and includes interviews with twenty to thirty key informants, observations of planning, estimation, review, and approval meetings at multiple hierarchical levels, as well as process documents, conversations with the company liaisons, and electronic records. Functional requirements, which are represented as “user stories”, along with testing requirements, are stored in a vertical information system. Multiple views of this data are available. Interviewees come from across the breadth of the involved processes, and included (or will include) vice presidents, project and product managers, business analysts, architects, software development managers, as well as those in research and development and quality assurance roles. This broad view of the company uses multiple rich sources across the complete release cycle. It provides the perspectives of multiple stakeholders necessary when viewing the internal boundaries of an organization, and it affords triangulation of findings to enhance reliability.

**Analysis**

Researchers will identify indications of uncertainty within the data, and code them using the research lenses described previously (Table 1). Specifically, the work design strategies as simplified by Sinha and Van de Ven (2005), the type of uncertainty using the categories of Mathiassen, et al. (2007), namely, identity, volatility and complexity, as well as aspects of the travelling metaphor will be coded. We use uncertainty as an indicator of the difficulty actors encounter in attempting to accomplish their tasks within the organizational structure. Further, requirements identified by interviews and project documents will be examined, such that the series of events that led to a resolution of task uncertainty over the life of the project can be explicitly described. These strings of events will be coded to reflect the process progression and organizational structures involved, to best identify which patterns emerge.

Models and storylines developed by pattern-matching and explanation-building (Yin 2009) of aggregated and coded will then be subject to verification and attempted disconfirmation through triangulation of the multiple available sources of data.

Due to the timeline of the research and the volume of qualitative data to be analyzed, it is too early to provide sufficient analysis, but several patterns described by Mintzberg (1993) are facially evident, including the way GridCo uses specific roles as knowledge leaders, and as a consequence creates slack resources (Galbraith 1973) while seeking standardized skills in a professional core that works independently from colleagues (Mintzberg 1993). Process reinforcers and process orchestrators, what Mintzberg (1993) refers to as “technostructure” are also strongly evident, as GridCo is a very process-driven organization. Lastly, there is a clear investment in vertical information systems, which is predicted by Galbraith (1973) to reduce uncertainty by increasing information processing capacity of the organization, a notion validated by multiple information systems researchers. However, despite a vertical information system, much of the data it provides is duplicated in multiple ways during administration and coordination activities. For example, text stored in the system is often read aloud during coordination meetings. The reasons for this behavior are being explored during data collection, and will be one of several items of focus during analysis.
Table 1: Framework for Analyzing Traveling of Requirements

<table>
<thead>
<tr>
<th>Theory</th>
<th>Code</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Uncertainty</td>
<td>Identity</td>
<td>Difficulty in the knowing of requirements caused by communications gaps.</td>
</tr>
<tr>
<td>(Mathiassen et al. 2007)</td>
<td>Volatility</td>
<td>The changing of requirements whether for internal or external reasons (e.g., time, budget, changing market or customer preferences).</td>
</tr>
<tr>
<td></td>
<td>Complexity</td>
<td>Difficulty in specifying and communicating requirements; includes the cognitive load required to understand the effects of implementation.</td>
</tr>
<tr>
<td>Work Design</td>
<td>Horizontal</td>
<td>Modular or serial work design. May involve mutual adjustment.</td>
</tr>
<tr>
<td>Contingency Theory</td>
<td>Share</td>
<td>The movement of an idea across a boundary. Encompasses changes that occur due to interpretations.</td>
</tr>
<tr>
<td>(Galbraith 1973)</td>
<td>Translate</td>
<td>The (re)enacting or materializing of an idea in a different form, using a different syntax.</td>
</tr>
<tr>
<td>Boundaries</td>
<td>Construct</td>
<td>The explication of an idea within a given syntactic/semantic context.</td>
</tr>
<tr>
<td>(Carlile 2002)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Discussion

Although significant discussion before completing rigorous analysis is premature, some themes have been observed that indicate the directions and possible contributions of this research.

First, is the whiplash effect of requirement dependencies. Even though GridCo employs risk models to provide management some confidence in cost–benefit and risk–rewards analyses, their models may require revision. Software projects, particularly work on the centralized command-and-control system rely on hardware (and its associated firmware) being completed to a particular level before work can begin. This means the work on related software requirements—which reside in a different project than the hardware component—is subject to not only to its standard risk variance, but also to the sum of all risks of the dependent project. These project and functional boundaries within the organization are reinforced by existing project processes, which have led some within the organization to call for a more holistic management of the project portfolio, and exploration of ways to span these boundaries with a pragmatic approach. In particular, hardware-dependent requirements appear to pose greater task uncertainty, in part due to the significantly greater development and manufacturing time requirements.

Second, requirements uncertainties, in identity, volatility and complexity are evident throughout the release cycle. Although the organization’s processes serve to enable and reinforce coordination through structural boundary spanning (particularly along the hierarchy), at times it also impedes the success of the organization as some of these processes are ill-adapted to the uncertain nature of software, resulting in anomalies such as ex post facto approval of changes to project scope, schedule and budget, or the beginning of software development before requirements are accepted into a project. However, these events where structure is ill-fitting may represent acceptable costs when compared with the added...
complexity of utilizing different processes for different project types within the same organization. This question is echoed by Child (1977), who asks whether an organization should “set a limit on its internal formalization in order to remain adaptable, or should it allow this to rise as a means of coping administratively with the internal complexity that tends to accompany large scale?” (p. 175).

Third, although not wholly related to the travelling of requirements, the structure of development on maintenance issues may limit the effects of double-loop learning (Nerur and Balijepally 2007) predicted by theory. GridCo separates a cadre of four development teams on a six-month rotation focused on maintenance issues. This has the advantage of providing a more predictive level of staffing for project issues, but at the cost of more real-time learning at the team level.

Conclusion

This research-in-progress provides a novel look at structure and processes in software organizations from the perspective of requirements. Not only will this research provide insight into the travelling and translation of requirements in development of packaged software, travelling of requirements represents a useful tool for mapping horizontal and vertical boundaries of an organization.

References


SWEBOK. 2013. "Guide to the Software Engineering Body of Knowledge (SWEBOK V3)." IEEE.


