Chains of Control in Agile Software Development

Completed Research Paper

Thomas A. Gregory
Georgia State University
tom.gregory@ceprin.org

Chaitanya Sambhara
Georgia State University
chaitanya.sambhara@ceprin.org

Lars Mathiassen
Georgia State University
lars.mathiassen@ceprin.org

ABSTRACT

Although extant literature on control in software development provides interesting and rich insights, it also indicates existing constructs are, in some ways, inadequate as intellectual means to support investigation of control practices. Most importantly, current theorizing distinguishes between two types of formal controls (outcome and behavior) and two types of informal controls (clan and self), making it impossible to investigate informal outcome and behavior controls and formal clan and self-controls. We therefore introduce a deconstructed framework for describing management controls and apply it to a holistic examination of controls in a case study of a multi-site agile software development organization. The empirical examination revealed consequent controls across a process and multiple representations of control across hierarchical levels of analysis. As a result, we offer a refined, deconstructed framework for investigating control practices, and the concept of chains of control as a useful tool for examining control practices in software development.

Keywords

Control theory, agile, software development, case study.

INTRODUCTION

The agile manifesto, in its purest form, deemphasizes formal processes and tools in favor of individuals and interactions (Fowler and Highsmith 2001). Yet, in corporate settings, the need to manage risks and balance the competing interests of stakeholders lead to the central dynamic of software development: the trade-off between control and flexibility (Harris, Collins and Hevner 2009). The idea of controls seems anathema to agile practitioners at first blush, yet controls are, in part, what distinguish flexible agile development processes from ad hoc ones. Most organizations that self-describe as agile adopt only certain practices. As a result, it is not clear what constitute “pure” agile practices (Baskerville, Pries-Heje and Madsen 2011). Still, the study of agile software development as it exists in organizations is of interest to information systems researchers. Further, as an organizational context implies controls, investigating how control is practiced in an agile software development may lead to insights regarding both controls and agile development. Just as important is investigating whether extant control theory adequately describes controls in agile development processes. To achieve these joint ends, this paper embarks on an in-depth, empirical examination of control practices within an agile software development firm that develops and sells a single product: a marketing automation platform.

The paper addresses multiple gaps in extant research. The setting is unusual, as this case examines a full spectrum of controls in a large, distributed, mono-product software organization with a platform of products. Whereas previous research is typically retrospective and focused on vendor-client dyads (e.g., Kirsch, Sambamurthy, Ko and Purvis 2002), this study observed and investigated controls as they occurred. This focus on the internals of an organization permits a more in-depth examination of social and individual-level effects implicated in agile control practices. Further, the interactions between coexisting formal and informal controls is rarely discussed (Tiwana 2010) and our empirical material afforded such a perspective. Lastly, most studies of controls in software development have a project-level focus. In contrast, our study takes a holistic view of the product development organization. In addition to addressing these gaps, this study provides important theoretical contributions: (1) a refined, deconstructed framework for investigating control practices, and (2) the concept of chains of control as a useful tool for examining control practices in software development.

BACKGROUND

The theory of organizational controls was conceived to explain the governance mechanisms established between two parties known as a controller and a controlee. Accordingly, our conceptualization of control aligns with Kirsch (1997, p. 217): “control is exercised via mechanisms—devices used by controllers to ensure proper controlee behavior—which, when successfully implemented, result in the regulation of behavior.” In order to align controlee’s actions and behavior towards its
objectives, the controller implements control mechanisms on the controlee (Eisenhardt 1985; Ouchi 1979). These control mechanisms are classified into two forms; formal and informal controls (Figure 1). Formal controls are implemented to estimate or measure the controlee’s adherence to the performance standards and prescribed behaviors. (Eisenhardt 1985; Ouchi 1979). Formal controls are historically conceptualized as being a performance evaluation strategy (Eisenhardt 1985; Kirsch 1997), or being connected to rewards (Maruping, Venkatesh and Agarwal 2009), although this part of its conceptualization is sometimes overlooked, particularly during operationalization of empirical work. Formal controls are sometimes simply described as those that are documented (e.g., Kirsch 2004; Maruping et al. 2009). The formality of evaluation mechanisms is generally not discussed. Formal controls are further classified as outcome and behavior control. Outcome controls are characterized by the documented specification of the desired outputs and goals of the organization, irrespective of the methods and procedures used by the controlee to achieve them, while behavior controls are characterized by the controlee’s adherence to the prescribed methods and documented procedures (Eisenhardt 1985; Ouchi 1979). Informal controls are characterized by their emphasis on social and cultural norms, and are further split into clan and self-control. Clan controls are used to align controlee’s goals to that of controller’s by assimilating the controlee into the organization’s social norms that include shared values, common beliefs and customs (Eisenhardt 1985; Ouchi 1979). Self-control refers to the inter-personal form of control that relies on the controlee’s adherence to the actions and behaviors that are best aligned to those of controller’s interests without external supervision (Kirsch 1997; Tiwana and Keil 2009). Self-control may be encouraged and enabled by the imposition of workplace structures (Kirsch et al. 2002).

The optimal choice for these control mechanisms is determined by the two important characteristics of the assigned task: a) degree of outcome measurability and b) degree of programmability. When outcome measurability is high but task programmability is low, use of outcome control is considered the optimal choice. When the outcome measurability is low and task programmability is high, behavior control is considered the optimal choice. When both characteristics of the assigned task are high, either of the formal control mechanisms may be implemented. When both characteristics are low, use of informal control is recommended.

While the impact of control mechanisms and their effectiveness in achieving organizational objectives has been explored in strategy and management literature, information systems literature has played a key role in understanding how overlapping and redundant control mechanisms may be combined to maximize their effectiveness. Software development literature has demonstrated specific combinations of controls are more effective in enhancing critical organizational competencies (e.g., ambidexterity) (Tiwana 2010). These combinations form a unique portfolio of controls that helps organizations improve process governance (Choudhury and Sabherwal 2003; Kirsch 1997).

While the literature has explored both formal outcome and formal behavior controls, investigation of informal control has primarily been restricted clan control. The combination of clan control along with the other formal controls has also been explored to some extent (Kirsch et al. 2002; Tiwana 2010). Though some studies have explored the possibility of self-control as a viable control mechanism (Henderson and Lee 1992; Kirsch et al. 2002; Maruping et al. 2009) their effectiveness when used in combination with other control mechanisms, particularly clan control is not well understood. Although some studies explore the combination of self-control with formal outcome controls (for example see Maruping et al. 2009), these studies provide only limited insight into the relationship between self-control and clan control, or examine only one form of informal control.

Situated in a single organization, the objective of this case study is to examine control practices in software development with an open mind to how combinations of controls play out in real world organizations. With this objective in mind, we observe the extant conceptualization of control (Figure 1) excludes consideration of certain types of control, e.g. formal clan control or informal output control that indeed may play key roles on software practice. To frame our empirical investigation, we therefore start by deconstructing the known conceptualization of control to form a simple, descriptive framework for investigating how different forms of control may combine together to form practices of control in software development.

DECONSTRUCTING CONTROL

The characterization of controls in the extant literature is based on the assertion that controls should be classified either as formal or informal controls. Outcome or behavior controls are, by conventional definition, formal (Eisenhardt 1985; Ouchi 1979). Similarly, informal controls are either clan or individually based. As a result, formal controls are considered to be orthogonal to informal controls with two distinct types of each. However, as (Kirsch 2004 p. 375) notes, there is disagreement among researchers about this conceptualization of controls, and explains that some researchers argue that clan
control is a form of behavioral control (e.g., Govindarajan and Fisher 1990; c.f., Kirsch et al. 2002). This is logical, as Ouchi (1979) describes clan control as alignment of goals (outcomes) and behaviors with social norms. As one of several examples, Ouchi describes the promotion of certain employees as a way of expressing approval for particular outcomes or behaviors. If clan control is a form of behavior control, informal controls may thus have traits of formal control. Although Kirsch (2004) finds differing effects for behavior and clan controls, this could be because of their differing levels of formality or controls addressing different types of behavior. Moreover, while formal behavior and formal outcome are easy to define, research has not moved beyond obviously formal controls to identify behavior and outcome characteristics in informal controls, having been thus far content with the broad label of clan control. This may be, in part, because the traditional separation of formal from informal has led to informal controls becoming a “catch all” category for some researchers. For example, Maruping, Venkatesh, and Agarwal (2009) suggested management might encourage self-monitoring behavior via incentives, simply dubbing these as informal and having aspects of clan and self controls, without fully categorizing incentives as clan or self controls. Such incentives are formal rewards for an outcome favorable to the controller, but is intended to modify social norms (clan), and promote self monitoring (behavior), all through a single controller-driven mechanism. Another concern with the traditional characterization of control is that formalization in software development is arguably best conceptualized as a continuum with degrees of formality and informality (Mathiassen and Munk-Madsen 1986). For these reasons, extant constructs for describing controls are inadequate, confirming the need for deconstruction of controls.

In addition to minor field disagreement and weakening of the formal–informal classification, the strength of clan control raises questions of formality. Clans impose rituals and values, and inculcate shared beliefs, which new entrants are expected to uphold. Clan members are controlees of the clan, and are expected to align their goals and behaviors with the clan (Ouchi 1979). Clan rituals, however, are not necessarily documented, and hence not classified as formal controls, despite possible repetition and embeddedness. Clan-driven processes repeated sufficiently often become de facto formal structures; this indicates formal–informal conceptualization should be separate from determinations of whether the control stems from management or social influence.

The common conceptualization of clan poses another difficulty, as the notion of clan control leaves the question of level of analysis indistinct. If social controls exist only informally, how, then can an individual’s cognizance of social control be confirmed? At the level of the controlee, desired behavior (or outcome) might occur because of pressure to conform to social norms, or from self-regulating behavior, which leads to level of analysis concerns. Control is known to be a multi-dimensional construct (Kirsch 2004), which can make classification problematic. Teasing apart the distinction between group and individual effects has been addressed repeatedly in organizational literature, which should be taken advantage of when considering organizational controls.

To deconstruct extant control concepts, we distinguish between three dimensions of controls: the method of control expressed through the degree of formality or informalitity, the focus of control distinguishing between behavior or outcome control, and, finally the level of control as illustrated in Figure 2. These dimensions are conceptualized as being orthogonal to each other, rather than nested subcategories as described in extant literature (Figure 1). To capture differences between clan and self-control, and since controls can operate both in hierarchies (Ouchi 1979, p. 838) and at a project level (Tiwana et al. 2009), we distinguish between individual, team and organization levels of control. The strengths of this deconstructed conceptualization is that it covers the different forms of control in extant control theory without making any assumptions about how variations along the three dimensions of control may combine in practice.

**RESEARCH DESIGN**

**Method**

This research adopts an interpretive case study methodology. Case studies are effective mechanisms to study organizational controls, as controls depend greatly on contextual factors (Eisenhardt and Martin 2000). Moreover, case studies are appropriate methods when faced with “how” and “why” questions, particularly in “real life” contexts (Yin 2009) and bring nuance, depth and roundedness to complex data (Mason 2007). Research was conducted over a period of thirteen months during a close collaborative practice relationship with the company (Mathiassen 2002). Researchers adopted an engaged
scholarship approach, which is “a participative form of research for obtaining the advice and perspectives of key stakeholders … to understand a complex social problem” (Van de Ven 2007).

Data
Twenty-one semi-structured interviews were conducted across all levels of the company from developer to executive. Most interviewees worked primarily from the company’s headquarters (Eastern U.S.), but some worked primarily at a remote development office (Western U.S.). Interviews of remote employees were typically conducted via videoconference (a regular practice at this company), but some remote employees were available to interview while visiting the company’s primary site. Interviewees represented diverse functions related to the development process, including members of agile development teams, customer support, network operations, release management, and QA, and were selected on a theoretical basis to achieve multiple points of triangulation and saturation. Because of their diverse roles, questions necessarily differed somewhat, but in all cases focused on the subject’s perspective of the project and software development organizations. Interviews were semi-structured to explore formal and informal controls, and how those affected behavior and outcomes across organizational levels. The list of interviewees is summarized in Table 1.

In addition to interviews, researchers observed development team meetings, participated throughout the research period in in management-level meetings regarding software controls (typically bi-weekly), and were provided access to process documents and measures of development productivity. Additionally, follow-up interviews and email exchanges were performed at the team and management level to enable iterative discovery and confirm analysis as theoretical explanations of control practices became clearer. This plethora of data sources (Table 1) permitted triangulation of multiple perspectives, and provided a rich basis for theorizing.

Setting
Research occurred at a medium-sized international marketing software-as-a-service (SAAS) firm, Marmot (a pseudonym). Marmot produces an online platform product delivering marketing automation and customer management services to clients across the globe. Marmot uses agile software development methodologies, accomplished by seven development teams via two-week sprints culminating in approximately three releases a year. Agile processes have been in place at Marmot for approximately five years.

As with many software companies that describe themselves as agile (Baskerville et al. 2011), the product processes

<table>
<thead>
<tr>
<th>Data source</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee Interviews</td>
<td>21 semi-structured interviews:</td>
</tr>
<tr>
<td></td>
<td>• 2 C-Level Executives (CEO &amp; CFO)</td>
</tr>
<tr>
<td></td>
<td>• 2 Vice Presidents (Product Development &amp; Marketing)</td>
</tr>
<tr>
<td></td>
<td>• 3 Directors (QA, Development, Network Operations)</td>
</tr>
<tr>
<td></td>
<td>• 4 Operations &amp; Support Managers</td>
</tr>
<tr>
<td></td>
<td>• 7 Team Members (Software Architect, Tech Lead (Lead Developer), 3 Developers, Quality Assurance, Business Analyst)</td>
</tr>
<tr>
<td></td>
<td>• Product Manager</td>
</tr>
<tr>
<td></td>
<td>• 3 Other Managers (Release, Configuration, QA)</td>
</tr>
<tr>
<td>Meeting recordings</td>
<td>Audio recordings from regular “workshop” meetings with management group, as well as recordings of two development team meetings.</td>
</tr>
<tr>
<td>Emails</td>
<td>Written follow-up from meetings and interviews.</td>
</tr>
<tr>
<td>Team observations</td>
<td>Observation of team “grooming” (estimation) and “retrospective”</td>
</tr>
<tr>
<td>Personal meeting notes</td>
<td>Researcher’s written notes and reactions to management meetings and interviews.</td>
</tr>
<tr>
<td>Company documents</td>
<td>Company’s process documents, samples of weekly metrics, reports by employees listing formal controls used in the company.</td>
</tr>
<tr>
<td>Dialectic reflection</td>
<td>Post-meeting notes and audio recordings of researchers’ thoughts, questions, and challenges to each other’s thinking.</td>
</tr>
<tr>
<td>Automated metrics</td>
<td>Marmot maintains automated metrics for its agile software teams, including story points attempted/completed, defects found “in the wild”, bugs squashed, test coverage, etc.</td>
</tr>
</tbody>
</table>

Table 1: Data Sources
governing the software development function implemented a more “waterfall”-style, plan-driven process that fed requirements to the development teams, which some authors refer to as “waterfall-up-front” (e.g., Sliger 2006). The product management processes were somewhat loosely defined and in flux during the observation period (trending toward increased formality and structure). Following a series of sprints, the development teams moved to a regression testing, hardening, and release phase that is also typical of plan-based organizations (“waterfall-at-end”). This mixture of agile and waterfall layers was referred to by some at the company as “scrummerfall”. Development teams could theoretically be tasked with development of any part of the software platform product, but in practice tended to have focused specialties. Teams were cross-functional, and included an architect, developers and QA personnel. Experienced personnel from across several teams formed additional special action groups (e.g., interface designers, architects, process improvement), which served to increase inter-team coordination and resulted in a high level of coordination within the product group.

The organizational culture was firmly in favor of team autonomy and team-level recognition. Teams were given their own budgets for activities and rewards, with the possibility of increases as team-level incentives. Each team had a name, with a labeled and custom-designed workspace, and were referred to by their chosen name throughout the organization. Team autonomy extended to governance of the teams’ internal processes. Most used a variation of SCRUM; one team adopted Kanban. However, this autonomy came at a loss of control by managers external to the teams. Respondents consistently mentioned variations in development and reporting practices between teams, and cited a handful of failed initiatives, such as data collection and metrics practices implemented by management, as evidence of the firm’s emphasis on teams. Accordingly, during the observation period there was a slow movement toward increased management control at the expense of team and individual autonomy.

ANALYSIS

Drawing on the deconstructed framework of control (Figure 2) we identified manifestations of control as evidenced across various development activities. Accordingly, we compiled a list of control mechanisms at Marmot based on interviews, observations, and other data. A partial summary of this list is represented in Table 2. Data regarding controls manifesting in development activities were tagged according to the deconstructed framework to create rich descriptions of control practices, which were discussed and refined by the research team based on evidence. Analysis was performed concurrent with data collection, which permitted later interviews to focus on gaps in data coverage, which typically occurred at different levels of analysis. The development activities and related control practices discussed in this paper were selected to represent the situation at Marmot, to focus on agile software principles, as well as for theoretical coverage. For space reasons, only two of these controls are examined in this paper: user stories and unit testing. These are selected because they contribute to theory in contrasting ways.

Control Related to User Stories

User stories are, on their face, formal controls: they are a documented set of requirements (goals) to be achieved by development. In the case of Marmot, the stories are stored, shared, and transformed as an artifact via an information system. Stories are the mechanism employed by the organization to achieve its strategic software goals. They are documentation of
<table>
<thead>
<tr>
<th>Development Activity</th>
<th>Evidence of Control</th>
<th>Consequences</th>
</tr>
</thead>
</table>
| User Stories         | • Documented set of short-term organization-level goals for software development (*formal, outcome*) | • Leads to additional controls: Publishing progress toward outcome (e.g., burndown charts) creates additional intermediate behavior & outcome controls targeting the team level  
• Stories translated to functional and acceptance tests (additional outcome controls)  
• Team-level awareness of relative productivity  
• Development necessitates unit tests |
| Progress (“burndown”) charting | • Teams’ progress toward sprint shared with organization (*formal, outcome, organization*)  
• Shared status creates team-level pressures to produce on schedule (*team*) | • Team members wary of any metric that could be used to compare teams  
• Multiple subjects expressed the view that teams’ self-evaluations were more important than inter-team comparisons |
| Unit testing         | • A required deliverable (*organization*)  
• Code must pass test (*formal, outcome*)  
• Team pressure to delivery timely tests (*team*)  
• Developers write own unit tests (*self*) | • Test are often created hastily (on a deadline), sometimes even after code complete deadline (the schedule is an implicit behavioral control)  
• Because of deadline pressures, sometimes tests are wrested rather than redesigning (*adverse behavior control*)  
• Use of unit tests reinforces test-driven development (TDD) as a preferred practice (*behavior, self*) |
| Code Reviewing       | • Typically ad hoc, within team (*informal, team*) | • Infrequent practice at Marmot  
• Complements other control practices |
| Defect reporting     | • Temporally distant feedback of initial quality (*formal, outcome*)  
• When possible, assigned to the teams/persons that introduced the defect (*formal, team, self*)  
• Developers are expected to learn from mistakes (*self*) | • Temporal distance limits local responsiveness, but extends visibility to multiple levels of the organization (*organization*)  
• Assigned to the “causing” team to encourage learning (*behavior*)  
• Leads to regression tests (*outcome*)  
• Teams attempt to avoid bugs through review and training (*informal, team*)  
• Defects are not publicly counted by team to avoid organization pressure (an explicit avoidance of control) |
| Retrospectives (Team-level reviews) | • *Ex post* qualitative outcome/behavioral control, formalized by meeting  
• Team meeting strengthens socialization (*team, self*) | • Social rituals raise awareness of norms (*self*)  
• Opportunity for positional leaders (Team Lead, Architect, QA Lead, Delivery Manager) to shift the attention of the team or reinforce organization-level expectations |
| Practice Guides      | • Documented organization-level control targets both outcome and behavior (*formal*) | • Organization-level practice influences team-level expectations of quality  
• Self-control due to documented expectation, weakened or strengthened by team norms (*team*) |
| Book clubs           | • Organization-level social pressure to participate (*organization, behavior*)  
• Club-level pressure to participate after joining (*team, behavior*)  
• Personal application of concepts (*self, behavior*) | • Culture of learning influences expectations, socialization, and personal expectations (*team, self, behavior*)  
• Indistinct effect on product outcomes  
• Greater interaction leads to stronger inter-team relationships (*team*) |

**Table 2: Sample Activities and Control Practices at Marmot**
desired outcomes, and as such, almost perfectly align with the definition of formal controls. Stories are built from a list of backlog items, which might be conceptualized as “potential controls”, which are formally defined, and also exist as an information system artifact. However, stories are inefficient evaluation mechanisms. An organization concerned with validating development output with stories—as Marmot is—instincts translations of those story requirements as unit and functional tests, which also serve as formal outcome controls. Progress toward story completion (and goal achievement) is tracked and shared via a published burndown chart, which thus serves as an additional control mechanism. Its effect, though, in addition to being a measure of outcome, is to focus developer efforts on clan progress, and align focus on behaviors that lead to appropriate outcomes. Any schedule or representation of progress against temporal goals would have this effect. Thus, the introduction of user stories in an agile development process leads to consequent incidence of additional controls through stages of the development process, and many of these controls are translations of the user story requirements (Figure 3).

Control Related to Unit Tests

Unit tests are an interesting example of control, and demonstrate why clear exploration of perspective is important (Figure 4). When considered from the organizational level (Marmot’s product group), use of unit tests is required. Unit tests are stored (documented) as artifacts in an information system. These are characteristics of formality (documentation, and use as evaluative tool), and thus unit tests are formal controls. Unit tests at Marmot primarily aim to evaluate the quality of the produced code, and are a representation of project goals, characteristics of outcome control. However, unit test are not used (formally) in the evaluation of the developer, but creating and applying unit tests are required for developers, which indicates behavior control. Thus, when considered in light of the work product, unit tests are outcome controls; when considered in relation to the development process, unit tests are behavior controls.

Unit tests also have a side effect at the team level where the existence and use of unit tests supports a culture of Test-Driven Development (TDD), particularly among senior developers. This informal social pressure acts as a behavior control (albeit weakly), and is a consequence of the existence of a formal control at the organizational level.

As a formal control and artifact in an information system, unit tests permeate organizational levels. Teams and individuals are expected to comply with the organizational mandate of unit testing. Uniquely, the formal expression of unit tests as a control extends to the level of individual control, as developers are expected to write their own unit tests. The act of writing the test is a behavior control, and the completed unit test serves as an outcome control for the developer’s own code. Thus, developers write a translated expression of the standards used to evaluate their own code. Despite its formality, the control is weaker at this level than some in the organization would prefer. Many times, developers feel rushed, and consequentiality the quality of unit tests is degraded. For example, one subject indicated that occasionally developers submitted their unit tests after the code deadline by which code had already been submitted. This indicates only a weak initial effect on output quality, although once created, the unit tests are reused. As an aside, the test suite at Marmot has become so large it is inefficient to maintain. (Initiatives at the company to reduce the size of the test suite without impacting its effectiveness are underway.) Lastly, those developers who chose to adopt TDD practices impose on themselves an informal behavior control. Thus, at the individual control level, unit tests demonstrate characteristics of formality and informality as well as both outcome and behavior control.

DISCUSSION

This research seeks to further develop our understanding of how control is practiced in software development. Current control constructs (Figure 1) offer rich support for understanding how different control mechanisms may be applied to manage software development in real world organizations. However, although literature provides interesting and rich insights into how control is represented in software development, they also indicate that the existing constructs are, in some ways, inadequate as intellectual means to support investigation of control practices. Most importantly, current theorizing distinguishes between two types of formal controls (outcome and behavior) and two types of informal controls (clan and self), making it impossible to capture control practices based on informal outcome and behavior controls or clan and self-controls that may present with characteristics of formality.
Against this backdrop we deconstructed existing control constructs to develop a new framework for investigating control practices in software development (Figure 2). We then applied this framework to rich data from agile development in Marmot to reveal how control was practiced related to specific activities across the development process. This led to two quite different and complementary exemplary analyses, one focused on control related to user stories and another focused on control related to unit tests. By focusing on how control unfolds in relation to development activities, rather than emphasizing the manifestation of different forms of control mechanisms, we arrived at a new understanding of how software development is managed and executed enabled by chains of control, i.e. connected instantiations of control that support specific activities in software development. The analysis of the user story control and its antecedent and consequent controls across a process, may be described as a horizontal control chain that help coordinate several interdependent activities across the software development process. In contrast, the examination of unit tests as control across multiple organizational levels revealed multiple representations of control, which may be described as a vertical control chain that support interaction between actors on different organizational levels in software development.

As a new approach to investigate control in software development, chains of control are consistent with the terms “horizontal” and “vertical” as used by Sinha and Va de Ven (2005) to describe work design across organizational units. Accordingly, vertical control chains address how resources, knowledge, and authority of a software development operation may be contained within one level of an organization, or may be divided among many hierarchical levels. In contrast, horizontal control chains address how activities may be distributed across many organizational units of a software development operation, where each provides a component or contribution to the overall software product.

The explication and conceptualization of these chains of control serve to demonstrate the validity of the deconstructed framework that views clan and self as levels of analysis, rather than as specific control types. Formal mechanisms of control, such as user stories, perhaps because of their concrete (documented) nature, crossed hierarchical boundaries and were represented similarly a different levels of analysis more consistently than informal controls. In general, informal controls tended to be based on events and structures, while formal controls were typically instantiated as IS artifacts. Tiwana (2010) asserted, “Informal control mechanisms strengthen the influence of formal behavior control mechanisms on systems development ambidexterity (complementary effects) but weaken the influence of formal outcome control mechanisms (substitutive effects).” His observation is consistent with unit tests as a control mechanism, in that there was a clear social and formal pressure to create unit tests (behavior control), and there was evidence suggesting unit tests served as a poor outcome control (in the short term), as they were too frequently rushed and sometimes completed after code had already been delivered. The notion of “emergent outcome controls” as introduced by Harris, et al., (2009) complements our description of horizontal chains of control. In the examination of user stories given above, the consequent controls emerged as translations of the user stories. Other emergent controls are likely to manifest along horizontal chains.

CONCLUSION

This paper has demonstrated a new way of looking at control in software development. The proposed framework refines the descriptive language of controls and is derived by deconstruction and subsequently validated within an empirical study. Using an organization focus, rather than the project focus typical of previous studies, we examined a full spectrum of controls in a large, distributed, mono-product software organization. Our empirical research identified chains of connected controls that exist horizontally across processes and vertically across organizational levels.

As a single-site case study, this research draws on insights in a particular context with particular practices, and particular controls. It may be that the controls discussed here would present differently in an organization with a different degree of implementation of agile practices, a different level of process maturity, or a different organizational culture. Case studies are useful for generalizing from description to theory (Lee and Baskerville 2003), as this paper does, but are not generalizable to other contexts. Additional exploration in other contexts is encouraged. Similarly, although the deconstruction performed here aligns with the data from the observed case, it may be that when applied to other contexts the deconstruction may need further elaboration. Lastly, the paper did not discuss in detail whether controls were emergent or deliberate, nor did it examine potential relationships between software architecture and process control or relationships between process structures and controls.

For the deconstructed framework presented here to be useful, it must necessarily be validated in additional contexts. Although informal and clan controls were examined here, they are necessarily context dependent, so future research should continue to explore the effects of organizations on control, and interactions between informal and formal controls in other contexts.
REFERENCES


