A Study of Enabling Factors for Rapid Fielding

Combined Practices to Balance Speed and Stability

Stephany Bellomo, Robert L. Nord, Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
sbellomo@sei.cmu.edu, rn@sei.cmu.edu, ozkaya@sei.cmu.edu

***Abstract*—Agile projects are showing greater promise in rapid fielding as compared to waterfall projects. However, there is a lack of clarity regarding what really constitutes and contributes to success. We interviewed project teams with incremental development lifecycles, from five government and commercial organizations, to gain a better understanding of success and failure factors for rapid fielding on their projects. A key area we explored involves how Agile projects deal with the pressure to rapidly deliver high-value capability, while maintaining project speed (delivering functionality to the users quickly) and product stability (providing reliable and flexible product architecture). For example, due to schedule pressure we often see a pattern of high initial velocity for weeks or months, followed by a slowing of velocity due to stability issues. Business stakeholders find this to be disruptive as the rate of capability delivery slows while the team addresses stability problems. We found that experienced practitioners, when faced with these challenges, do not apply Agile practices alone. Instead they combine practices—Agile, architecture, or other—in creative ways to respond quickly to unanticipated stability problems. In this paper, we summarize the practices practitioners we interviewed from Agile projects found most valuable and provide an overarching scenario that provides insight into how and why these practices emerge.**

***Index Terms*—agile software development, architecture, speed, stability, rapid fielding, software development practices**

I. Introduction

A commonly held view is that waterfall-based processes focusing on monolithic requirements, analysis, design, implementation and testing practices have led to the slowing of software delivery [1][7]. Because Agile projects show promise in improving speed, industry and government alike have been increasingly adopting Agile-based incremental software development practices. Many Agile success stories have been attributed to the adoption of practices such as increased team communication, collective ownership, frequent customer-visible releases, backlog-driven requirements management, continuous integration, and shorter iterations, but are these practices really the key enablers for rapid fielding? If they are, why do teams in highly regulated environments struggle as they adopt Agile, iterative, or hybrid methods? There is lack of clarity regarding which factors truly contribute to the ultimate goal of rapidly fielding tested software functionality to its intended end users [2] [3].

In order to better understand whether generalizable rapid-fielding success factors exist, we conducted an interview-driven study with five organizations that have adopted agile, iterative, software development practices from both government and commercial organizations. We spoke with Agile team members developing a variety of software systems, such as mission/business analysis support systems, COTS customization projects, software and hardware control systems, and simulators. The systems studied varied in length of operational use from pre-release to 14 years of production use.

Through these interviews, we observed that projects with a business goal of delivering capability rapidly must deal with a natural tension between the pressure to deliver functionality quickly (speed) and the desire for a reliable, stable, and flexible product (stability). We see evidence of this tension in our work with Agile projects; for example, we often see a pattern of high initial velocity for weeks or months due to schedule pressure followed by a slowing of velocity due to stability issues. This slowing in velocity negatively affects business stakeholders as the rate of capability delivery slows.

Speed and stability dimensions increasingly have been the subject of research interest. Speed and stability can be thought of as two ends of the rapid-fielding spectrum which can be useful for reasoning about architectural tradeoffs [4]. Martini et al. observed a tension between speed and reuse when agile and iterative practices were introduced to automotive organizations with established product line engineering practices [5]. They observe that increased reuse and increased speed are common competing business goals. In their work, architecture is identified as an enabler for reuse and Agile practices are recognized as enablers for achieving deployment speed.

During our interviews, we asked practitioners to describe examples of factors they believe enabled and inhibited speed and stability on their projects. We started the interviews with general definitions for speed and stability and then let the interviewees reshape the definitions as needed.

Our starting definition for speed was *enablers that promote rapid fielding.* Practitioners responded during interviews with examples of speed enablers from across a broad spectrum of development phases. For example, they gave examples of speed enablers related to the proof of concept, requirements, design, development, and testing phases. In addition, the definition of "rapidly" varied widely in examples practitioners

shared with us. Sometimes rapidly meant a day (e.g., an enabler that speeds up a daily build), weeks (e.g., an enabler that speeds up a sprint cycle), or even a few months or years (e.g., enablers that speed up initial proof of concept or approval for external release). We used the same approach for defining stability starting with a general definition letting practitioners tailor as desired.

Our starting definition for stability was *enablers that promote stability/flexibility in the software product.* Like the definitions for speed, the definitions of stability varied widely, though all stability-related definitions remained focused on architecture concerns. Some practitioners described stability in terms of the quality attributes of the product such as reliability, scalability, performance, security, etc. (quality attributes describe the qualities that stakeholders expect a system to provide [6]). Other practitioners described enablers for stability in terms of infrastructure investments. We did not scrutinize the definitions of speed and stability that practitioners provided. As long as reasonable rationale was given, accompanied by actual experiences, we considered the definitions reasonable examples of enablers and inhibitors.

Several insights emerged through the grounded-theory-based analysis method we applied. As we analyzed the interview data, we found that enabling practices fell into two types. The first type of enabling practices were the basic Agile practices commonly touted as contributors to the success of Agile projects, such as Scrum status meetings, continuous integration, test-driven development, etc.

A second type of enabling practices emerged when interviewees gave examples of how they addressed challenging situations. When practitioners were talking about addressing problems impacting their ability to rapidly field software, we observed that project teams didn't apply single Agile practices, architecture practices, or other practices. Rather, they often combined practices to address the problem in an incremental way. Experienced practitioners used their expertise to creatively combine practices from disciplines ranging from management to engineering to avoid significant disruptions in velocity. Some examples of these combined practices are release planning with architectural considerations, prototyping with quality attribute focus, release planning with external dependency management and test-driven development with quality attribute focus. We elaborate on several of these examples in Results, Section 3.

In addition, a common scenario also emerged through our interviews that gave us better insight into how these combined practices are used to balance speed and stability. Through this scenario we observe that a focus on speed often results in problems that trigger a focus on stability. In response, experienced practitioners combine Agile and architecture practices to address the problem. We refer to this pattern as the *Speed-triggers-stability* scenario.

We also explored inhibitors to rapid fielding. A number of inhibiting deficiencies and constraints emerged through the interviews; however, we noted a particularly high number of inhibiting factors associated with testing. Testing, certification, and accreditation are increasingly tagged as the most challenging sources of expenditure [1]. Four out of the five organizations we spoke with described situations where projects were not able to complete test cases within the targeted increment timeframe. Several practitioners we interviewed said this is due to increasing software complexity and limitations in expertise/tools on the project. Interviewees also shared several examples of delays resulting from slow-moving and incompatible enterprise processes, such as assurance certification, and hierarchical decision-making processes.

The rest of the paper is organized as follows. Section 2 describes our data collection and analysis approach. Section 3 summarizes our findings. Sections 4 and 5 discuss the implications of our results and conclude the paper, respectively.

II. Interview and Analysis Approach

We leveraged grounded-theory-based approach in our analysis. As we conducted interviews, we emulated Glaser's conceptual approach to grounded theory which aims to let the theory emerge from the data [9][13] while leveraging some of the structured steps described by Strauss [10][8]. The research design is described following this general flow:

- data collection
- developing memos and indicators
- coding (deriving concepts and categories)
- saturation and concept strength

A. Data Collection

We conducted each of the interviews, except for one, via teleconference. One interview was conducted per project. All of the interviews were recorded and transcribed. Each of the interviews lasted 60-120 minutes. The interviewees from each organization included technical and management staff (architects, developers, managers, and testers). We used a guiding question approach, in which we asked a general overarching question and then let the discussion flow naturally from there [9].

The guiding question we asked was *"What are factors that enable or inhibit rapid fielding on your project?"* We also asked interviewees to give examples of rapid-fielding enablers with respect to speed and stability. Sometimes the practitioners needed prompting to describe the speed and stability dimensions so we asked probing questions such as, *"What impact did the incident you described have on speed or stability?"* The organizational characteristics of the project teams we interviewed are summarized in Table 1, including the type of the system being developed, the iteration length and the approach used to manage releases.

The eight projects discussed in our interviews represented a variety of system types. Five of the systems were information processing systems (e.g., business analysis systems), two were COTS customization projects, one was a hardware controller, and one was a training simulator. The projects ranged from those in the inception phase to those with over a decade of production use. The product size ranged from 1 to 20 million software lines of code (SLOC) and team size ranged from teams of 5 to over 30 (team members included developers/ testers/ managers).

TABLE 1: ORGANIZATION CHARACTERISTICS

Project ID	Time in Production	Release Management Approach	Type	Product Size	Team Size	Sprint length / Prod Release Cycle
A-P1	Pre-release	Scrum	Case management system	<10M SLOC	10-20	2 weeks/ TBD
B-P1	12 years	Scrum	Analysis support system	<10M SLOC	10-20	2 weeks/ 6 months – 1 year
C-P1	3 years	Scrum	Training simulator	1-10M SLOC	>30	4–6 weeks/ 2–6 months
D-P1	Pre-release	Scrum	Enterprise information sharing portal	TBD	>30	2 weeks/ TBD
E-P1	12 years	Scrum	Doc management system	10-20M SLOC	9	2 weeks/ 1–3 months
E-P2	14 years	Incremental (prior to Scrum)	SQLWindows tool	<10M SLOC	10-15	N/A/ 1 year
E-P3	8 years	Incremental (prior to Scrum)	Hardware controller	<10M SLOC	5	2 weeks/ 2 months
E-P4	1.5 years	Scrum	Customization project of a packaged software system	10-20M SLOC	6	2 weeks/ 3 months

B. Memos and Indicators

Memoing is the first step in the analysis process. During memoing analysts begin to collect their thoughts in preparation for the full analysis phase (referred to as the "coding phase"). Memos are informal, written records of analysis [10]. For each transcribed interview, we went through the raw data, breaking the transcription at natural breaking points and creating memos. The data that researchers capture in memos are subject to their discretion [10]. We found that the most important data element we captured in memos was information indicating the possible presence of a concept. We adopted the use of the term indicator for the data element indicating presence of a concept in our memos (adapted from recent work by Adolph et al. [11]). We often captured an indicator of a concept as a snippet of raw data. An example of an indicator from our data is, statement, "We get a lot of value from weekly demos," which suggests support for the presence of the Prototyping concept. Indicators are important in the next step, in which we analyze data and identify concepts.

C. Coding (Deriving Concepts And Categories)

Coding is the analysis step. The primary objective of the coding step is to derive concepts from data. Concepts represent an analyst's understanding of what is being described through the examples of incidents. During the coding process, the analyst walks through the indicators generated from the raw data/memos deriving concepts [10]. The concepts are validated through the process of constant comparison in which the analyst goes through each incident in the data, comparing it to other incidents. Incidents found to be conceptually similar are grouped together and mapped to an emerging concept [10].

Through the constant comparison process, we derived rapid fielding enabler and inhibitor concepts as well as categories (see example in Fig. 1). Categories are higher level concepts used to relate or group lower level concepts [10]. Figure 1 shows the conceptual relationship between categories, concepts, and indicators and accompanying examples. The two key relationships represented in the diagram are: 1) indicators *suggest* concepts and 2) categories *contain (or group)* concepts. The example shows four indicators supporting the concept *prototyping with quality attributes,* which is part of the category, *practice*.

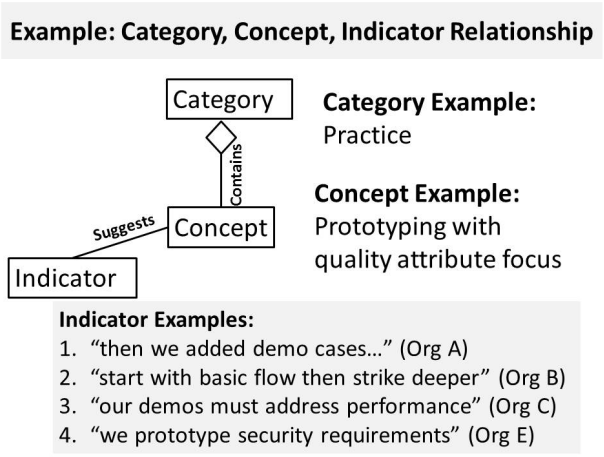

Fig. 1. Category, Concept, and Indicator Relationship (*adapted from [10]*)

D. Saturation and Concept Strength

Saturation is the process of acquiring sufficient data to develop each concept/category fully, in terms of its properties and dimensions, and to account for variation [10]. The goal of saturation is to gain confidence in emerging concepts and separate weaker concepts from stronger concepts. Our approach for identifying strong concepts in our data was to collect the data, methodically analyze concepts, and systematically calculate concept strength. We calculated concept strength by mapping the number of indicators to concepts. For example, in Fig. 1 the concept "Prototyping with quality attributes" is assigned concept strength of 4. Using this mechanism, we ordered concepts shown in Table, 2, Table 3, and Table 4. We counted each indicator identified in the data as an independent data element.

We also leveraged an approach used by Martini, et al. [5] for tagging the data with a speed/stability identifier. For example, the indicator, "We get a lot of value from weekly demos because we can incorporate user feedback more rapidly" was tagged as promoting speed. These speed/stability identifiers provided insight as to what interviewees perceive as the relationship between an enabler and speed and/or stability.

III. RESULTS

We begin this section with an overarching scenario that begins to shed light on how and why the practices described in interviews emerged. Using this scenario, we present the practice-related findings beginning with an overview of

enabling practices followed by several specific examples. We end the section with a summary of several key inhibitors and a brief discussion of the implications of inhibitor-related findings.

Before we discuss the overarching scenario, we introduce a unifying concept leveraged in the description of the scenario. The idea is that Agile project teams recognize that there is a desired software development state that enables them to quickly deliver releases that stakeholders value [4][12] (Fig. 2). When product development starts, this desired state has not yet been achieved. To achieve desired state, teams go through a Preparation phase focused on getting the infrastructure in place. This involves getting platforms and frameworks, as well as supporting tool environments, practices, processes, and team structures in place to support efficient and sustainable development of features. Once they have achieved the desired state, teams enter into a Preservation phase where the infrastructure is in place and they work to achieve a consistent velocity (avoiding major disruptions to speed). In this phase, the goal is to maintain balance. For example, it is important to neither over-optimize the supporting development infrastructure nor to quit working on it.

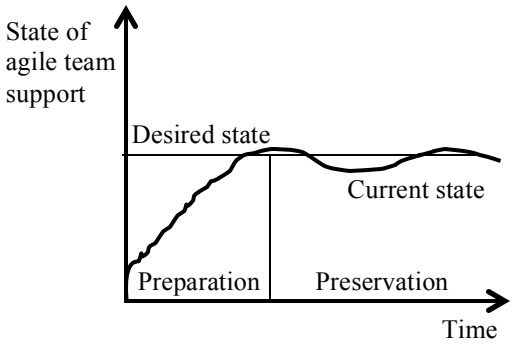

Fig. 2: Software development support for teams over time

As we spoke with organizations described in Table 1 in the Preparation state (A and D) and Preservation state (B, C, and E), a scenario emerged that illustrates how practitioners apply practices to stay within acceptable range of desired state. We refer to this as the Speed-triggers-stability scenario (Fig. 3).

We explain this scenario by walking through the steps S1-S4, illustrated in Fig 3. (S1) Due to business needs, there is significant pressure to field capability rapidly. We refer to this as a *Focus on speed*. We also note that at S1 the project is within acceptable tolerance of desired state. (S2) A stability problem occurs, such as embarrassingly poor system performance during a stakeholder demonstration. The problem puts the project outside acceptable tolerance of desired state triggering a focus on improving stability to get back into acceptable range. (S3) The project team responds to address the problem by applying a single practice or by combining practices. If the problem is visible (i.e., impacting delivery of needed capability) the team responds quickly and the resulting practice change is incorporated into the team's software development support structure without major project disruption. We refer to this as the *incremental response cycle*.

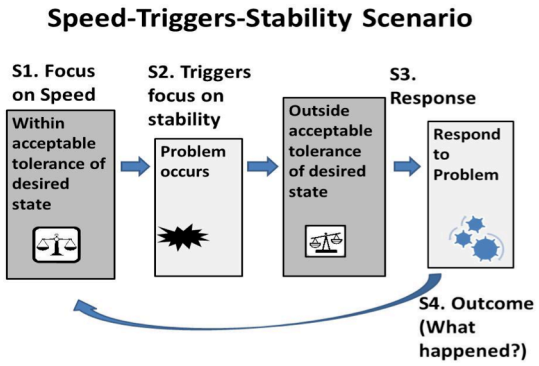

Fig. 3: Speed-triggers-stability scenario

If the problem is not visible (e.g., an architectural problem observable only by the development team), it may be difficult to make a strong case to expend development effort fixing it. Response is delayed and problems accumulate often requiring more effort later. We refer to this as the *big bang response cycle*. The big bang can result in significant disruption and effort. (S4) If the outcome of the response to the problem is good, we observe that it will bring the project back toward the acceptable tolerance of desired state for that particular problem.

A. Summary of Enablers

In this section, we summarize the enabling practice findings from our interviews, as summarized in Table 2.

TABLE 2: SUMMARY OF ENABLING PRACTICES: WITHIN ACCEPTABLE RANGE OF DESIRED STATE

SUMMARY OF ENABLING PRACTICES WITHIN ACCEPTABLE RANGE OF DESIRED STATE
• Vision Doc/roadmap (long-term release planning)
• Scrum collaborative management style
• Prototype/demo (community previews)
• External Dependency Management
• Use of collaborative tools foster communication
• Scrum status meeting
• Test-driven development
• Continuous integration
• Small dedicated team and limited scope
• Incremental release cycle
• End user involvement
• Evolutionary design and documentation
• Retrospective and periodic design reviews
• Use of standards and ref models
• Configuration Management
• Story points for productivity tracking
• Requirements to design traceability
• Proof of concept (for unproven tech)
• Pair programming

We define *practice* as a repeatable way of accomplishing an activity related to software product development or delivery; for example, we consider *prototyping* to validate requirements and gather user feedback a practice. We observe that enabling practices fell into two groups. The first group,

shown in Table 2, represents the set of practices that one would expect to find in any discussion with project teams about enabling Agile practices. These practices were typically described as enablers when projects were going well or "within acceptable range of desired state". The practices in the tables 2 and 3 are ordered by concept strength.

The next group of practices, shown in Table 3, emerged as practitioners gave examples describing how they dealt with problem situations where they were "outside of acceptable range of desired state". We noted that often practitioners would combine practices Agile and architecture practices in creative ways to address the problem. We identify these combined practices in bold font below.

TABLE 3: SUMMARY OF ENABLING PRACTICES: OUTSIDE OF ACCEPTABLE RANGE OF DESIRED STATE

SUMMARY OF ENABLING PRACTICES OUTSIDE OF ACCEPTABLE RANGE OF DESIRED STATE
• **Release planning with arch considerations**
• **Prototype/demo with quality attribute focus**
• **Release planning with Joint prioritization**
• **Test-driven development with quality attribute focus**
• **Dynamic organization and work assignment**
• **Release planning with legacy migration strategy**
• **Roadmap/Vision with external dependency mgmt**
• Root cause analysis to identify architecture issues
• **Dedicated team/specialized expertise for Tech Insertion**
• **Technical debt monitoring with quality attribute focus**
• Focus on strengthening infrastructure (runway)
• Retrospective and periodic design reviews
• Use of standards and ref models
• Backlog grooming
• Fault handling or performance monitoring
• **Vision document with architecture considerations**

B. Enabling Practice Examples

In this section, we describe some of the combined practices in more detail using the Speed-triggers-stability scenario. In most of these examples an Agile practice is in use when a problem pushes the team outside the acceptable range of desired state. The experienced practitioner augments the Agile practice with another practice to address the problem with minimal disruption to capability delivery.

1) Release Planning with Architecture Considerations: This practice extends the feature release planning process by adding architectural information to the feature description document prior to release prioritization. The example is provided by an architect from Organization C.

- ***S1. Focus on speed****:* The organization had adopted the Scrum release planning management process whereby the product owner prioritizes features ensuring a focus on speed [14]. After the backlog is prioritized, the product owner hands the prioritized backlog to the developer team to design and implement the features.
- ***S2. Triggers focus on stability****:* The trigger is the business moves from a centralized development model to a geographically distributed work model. An increasing focus on speed brings about the realization that teams need to work in parallel to meet schedule demands. The team eventually runs into challenges because there is not enough architectural definition in the feature documentation to allow the teams to "go off and work independently". This ultimately impacts release speed.
- ***S3. Response***: The team responds by augmenting the existing release planning practice. They attach a minimal design document containing architectural design information they called a "design memo" to the feature description document. Several considerations are taken into account as the design memos are developed including support for parallel development. Because this team responds quickly and incorporates this practice while continuing to deliver capability the example follows the incremental response cycle.
- ***S4. Outcome****:* By extending release planning with architecture information, the team was better able to identify tradeoffs to support parallel development. This practice was instrumental in enabling rapid development. In addition, the team noted that architectural changes were also made to promote modularity and support parallel development. We discuss this aspect in the subsection "Architecture change to promote stability."

Release planning with architecture considerations practice was widely supported. All the organizations we interviewed gave examples supporting this practice.

2) Prototyping/Demo with Quality Attribute Focus: This practice extends the prototyping/demo user feedback practice to include a focus on quality attributes. The example provided here comes from a project manager for Organization A. Note that project team members used the terms prototype and demo interchangeably.

- ***S1. Focus on speed****:* In this example, the team was under great pressure to deliver capability rapidly. Consequently, business stakeholders were very interested in seeing demonstrations of feature-related functionality (page layout, workflow, navigation, etc.) and less interested in demonstration of quality attribute-related requirements such as performance and security features.
- ***S2. Triggers focus on stability****:* Right before a pre-release milestone meeting system stakeholders began asking questions about scalability and performance. To gain an understanding of how well the system would respond under more strenuous conditions, they asked for a demonstration of system capability the team had planned to demonstrate but against a much larger data set than usual. Scalability had not been a design focus for the project team and, consequently, during the demonstration the users experienced an unacceptable drop in performance (response time was slow for some large searches).
- ***S3. Response****:* The visibility of these performance problems prompted the team to incorporate quality attribute considerations into their prototyping practice. This practice change was made with fairly minimal

disruption to the incremental release cycle. The team also did some refactoring to improve performance.

- ***S4. Outcome****:* As a result of this incident, the team incorporated performance and security-related scenarios to the demonstration suite. By extending prototyping to include these quality attribute concerns the team was able move back within an acceptable tolerance of desired state.

Organizations A, B, C, and E gave similar examples. Organization B said their prototyping process now begins with a demo of basic flow and they "strike deeper" to validate the design quality attribute requirements.

3) Roadmap/Vision with External Dependency Management: This practice incorporates external dependency analysis into the roadmap planning process to reduce the risk of being blind-sided by unanticipated external changes. These are dependencies outside of the team's sphere of control such as dependencies on expertise outside the team, infrastructure components governed by other parties, or difficult-to-reach users. Organization D provided the example below.

- ***S1. Focus on speed****:* A focus on speed led to limited focus on external dependency analysis. This put the team at risk for impacts by unmanaged external dependencies.
- ***S2. Triggers focus on stability***: The project was working aggressively on developing their first operational release. During an important development sprint, several firewall ports governed by an external party were closed without notice, causing sporadic and difficult-to-troubleshoot stability issues. Significant time was wasted targeting the source of this problem. Speed was impacted because effort was expended on troubleshooting the infrastructure problem rather than building features for the next sprint.
- ***S3. Response***: Since this problem was holding up development, the team took immediate action by analyzing dependencies and reassessing external dependency risks. Team members then came up with a mitigation strategy for each risk. Some mitigation strategies required modifications to the change management notification process and others required deeper understanding of dependencies on components being developed by other teams. The roadmap document, which contained a description of development by phases, was used to capture external dependency risks and mitigation strategies at the portfolio level. This change was incorporated into the ongoing practices with limited disruption to ongoing work; therefore, it followed the incremental response cycle. The team also adopted the practice of continuing to revisit external dependency analysis regularly to identify external dependency risks.
- ***S4. Outcome***: After updating the release plan documentation with external dependency information, the team experienced fewer instances of unanticipated port changes as well as other external changes.

Like Organization D, most of the organizations we interviewed said that they also had to manage external dependencies proactively. The Scrum Guide suggests that, to the extent possible, project teams should try limit external dependencies on other resources outside team to reduce the risk of unanticipated changes [14]. While the organizations we spoke with agreed philosophically with this idea, they said it is often not possible to avoid external dependencies due to the scale and interoperability requirements on their projects.

4) Test-Driven Development with Quality Attribute Focus: This practice merges test-driven practices, such as automated test-driven development and continuous integration, with a focus on runtime qualities such as performance, scalability, and security. This example comes from Organization E.

- ***S1. Focus on speed***: The team had developed a set of test cases that very effectively tested business functionality. However, they had a fixed deployment deadline and great schedule pressure so they did not have time to develop quality attribute-related test cases (in particular security-related test cases).
- ***S2. Triggers focus on stability***: Late in the development lifecycle, the team became nervous that the project software would not pass assurance testing and that late discovery of security vulnerabilities would cause them to miss the fixed deadline for deployment.
- ***S3. Response***: Team members responded by removing some of the planned features from the release refocusing the effort on shoring up security-related gaps. As they did this, they also incorporated additional security-related test cases into their regression test case suite. Because of the rapid response, and the continued focus on security after the incident, this example also followed the incremental response cycle path.
- ***S4. Outcome***: By extending test-driven development to incorporate security considerations (a quality attribute focus) the team was able to improve confidence that security requirements were addressed and avoid late discovery of schedule-impacting problems.

All the organizations we spoke with gave examples supporting this practice. We also noted that several of the organizations appear to be struggling to make their test activities fit into a rapid release cycle (particularly within a sprint). We discuss this issue further in the Inhibitors section.

5) Technical Debt Monitoring with Quality Attribute Focus: The metaphor of technical debt is used to refer to accumulating degradation of quality due to intentional and unintentional shortcuts [16]. During interviews we heard stories of problems due to unchecked technical debt leading to stability challenges and big bang response cycles. With this practice practitioners described steps they are taking to begin to put in place mechanisms to monitor technical debt. An architect with Organization B provided this example.

- ***S1. Focus on speed***: In order to speed up development time, the team purchased a COTS tool to enable team members to easily add new fields to web pages. This tool put in place a layer between the database and the application layers of the system. This appeared to be a

change that would promote stability by encapsulating other layers from the database layer.

- ***S2. Triggers focus on stability***: The problem is that now every field added to a web page through the tool creates a new XML-based query. Rather than having a manageable set of interfaces to the data layer to maintain there are many of these query-generated interfaces. As a result of this design decision, a change to the database schema may have an extensive ripple effect impacting many interfaces. In addition, making changes to the COTS tool requires a special skill set, so requests for changes queue up. So, what seemed like a positive change resulted in a negative impact to modifiability. While team members would like to change this situation, they have difficulty making a case for change because the problem is only visible to the development team. The development team lacks measures for communicating the impact of the problem to the business side.
- ***S3. Response***: Because the team can't easily make the problem visible to the business side, the development team is hoping to bundle this change with a future redesign effort - big bang response cycle style.
- ***S4. Outcome***: The outcome is that this problem still exists today and potentially impacts speed every time a new data element must be displayed on a page. The team is waiting for an opportunity to work the change in (or until the speed issue because too painful for the business).

This enabler was emphasized in interviews with organizations B and C. These teams were in the Preservation phase and had considerable experience working together. These teams described this type of issue as technical debt. Both projects described how, due to business pressure, they sometimes embed architectural change with unrelated features during feature development. This lack of transparency can result in incorrect productivity measures as well as unanticipated schedule impacts. Both projects described how they were in the early stages of working on ways to better measure and monitor technical debt. They expressed the belief that if they were able to make technical debt more visible to stakeholders, they could avoid the potentially costly and disruptive big bang cycle. As described in the example, unchecked technical debt can have consequences impacting quality attributes such as modifiability, therefore, we call this enabler technical debt monitoring with quality attribute focus.

6) Architectural Change to Promote Stability: This practice builds on an example described in the Release Planning with Architecture Considerations subsection. In this practice, project teams make architecture changes to address stability issues applying architecture tactics such as encapsulation, distributed design, layering, and so on to respond to stability issues. Due to the technical nature of this enabler, these examples were provided by technical staff such as developers and architects. We use an example from Organization C to describe this practice.

- ***S1. Focus on speed***: The team members explained they had been suffering from a monolithic software design for several years making even small changes time consuming and risky. The monolithic design also limited the team's ability to develop features in parallel. Although these problems impacted the effectiveness of the development team, the fact that the problems were not very visible to the business side caused design improvements to be put on the back burner for a long time.
- ***S2. Triggers focus on stability***: Adding the "design memo" practice improved the situation, however, the monolithic architecture continued to limit the team's effectiveness. Eventually the business stakeholders approved a redesign.
- ***S3. Response***: Due to difficulty in making this problem visible to the business stakeholders, this change along with others like it was extensively delayed and finally addressed through a major redesign. The response cycle was not incremental; it was big bang style.
- ***S4. Outcome***: Speed was impacted for a period of time as other planned features were put on hold during the redesign, however, after the redesign was finished the teams could more easily add new features and work in parallel (improving speed).

Organizations A, B, C, D and E all gave examples supporting this enabler. Based on our interviews, we see an association between the big bang response cycle and the lack of measures for technical debt. While changes to software resulting from the normal course of software evolution are expected in the sustainment phase [15], the problem with this scenario is there was not enough information to convince the business side to approve incremental architectural changes. While the business side members wouldn't agree to refactoring to fix the problem, they really did not like the big bang either. Projects B and C both said that their business stakeholders strongly dislike bug-fixing releases and major redesigns because of the impact on speed. So, what are experienced practitioners doing to avoid this? Organizations B and C said they are now starting to keep a list of design decisions that may cause technical debt to accumulate in the future. These projects suggest that improving visibility into technical debt, by keeping a list of design shortcuts and by other means, coupled with an incremental architectural change plan, could minimize the likelihood of a big bang response.

C. Summary of Inhibitors

In this subsection we summarize inhibitors to rapid fielding collected during our interviews. Table 4 lists the inhibitors ordered by concept strength. We focus the discussion primarily on concepts with high category strength or better shown in white portion of the table.

We found that major inhibitors to rapid fielding generally fell into categories of either constraints or practice deficiencies. As we analyzed the inhibitor data we saw relationships between some of the inhibitors (Table 4) and combined enabling practices (Table 3). For example, when practitioners

from Organizations B and C described incidents of applying the *Technical debt monitoring with quality attribute focus* enabler they also often mentioned these influencing constraints (inhibitors):

- *Desire for features limits requirements analysis or stability-related work*
- *Stability-related effort not entirely visible to business*
- *Limitations in measuring architectural technical debt*

TABLE 4: SUMMARY OF INHIBITORS

SUMMARY OF INHIBITORS
• Desire for features limits requirements analysis or stability-related work
• Slow business decision, feedback or review response time
• Problems due to challenges with external dependency management
• Stability-related effort not entirely visible to business
• Limitations in measuring architectural technical debt
• Inadequate analysis, design or proof-of-concept
• Inconsistent testing practices and/or deficiency in quality attribute focus
• Poor testing consistency
• Runway or infra limitations
• Resource limitations
• Poor configuration management limits reversibility
• Over-dependency on architect for architecture knowledge
• Selected COTS product limits flexibility
• Organizational standards limit design options
• Incompatible milestone lifecycles
• Business didn't buy into Scrum
• Arbitrary backlog grooming
• Personnel issues limit ability to track individual productivity

The focus on speed and difficulty making architectural problems visible to the business side often led to major redesigns or bug-fixing sprints (undesirable big bang response cycle). In these scenarios, technical debt builds until refactoring will no longer address the problem [16]. Another constraint, *Slow business decision, feedback, or review response time*, is also at the top of the list. Several organizations said that they wasted a lot of time waiting on important management decisions and lumbering enterprise certification processes over which they had no control. Organizations A, B, C, and D all gave examples of this. A question for future investigation may be "What role do these constraints play in inhibiting project teams from achieving desired state?"

We also noted several high-ranking inhibitors we categorized as practice deficiencies. For example, all of the organizations, except E, said that they were struggling with testing-related problems. Organization D struggled with developing test cases for complex and unpredictable functionality, such as user interaction, within a sprint or release cycle. We called this inhibitor, *Inconsistent testing practices and/or deficiency in quality attribute focus*. Several teams said they wanted to fully leverage Agile test-driven development practices; however, the team's testing expertise and tool knowledge was limited. These inefficiencies in testing practices often resulted in inconsistency in applying testing practices. Often teams said there just was not enough time to do all the testing they needed to do to produce the highest quality product. For example, Organization B acknowledged the need for performance and scalability testing on the project; however, because these tests take a lot of time when businesses pressure increases focus on quality attribute-related testing decreases. Organizations A and C both said their performance regression tests "take too long" so they conduct them when (and if) they can fit them in. We also note that there may be a relationship between this inhibitor and the high-ranking enabler *Testing practices with quality attribute focus*.

IV. DISCUSSION

In this section, we reflect on the approach and discuss ideas for future work. We found the structured nature of the grounded theory-based approach helped to organize the analysis process and ground our work in actual experiences from practitioners. The process of memoing from transcript-driven data limited interviewer bias by minimizing filtering that occurs during the note-taking process (although, of course, analyst biases still exist). We used a structured approach to build a lengthy, and somewhat complex, spreadsheet carefully mapping the data to concepts/categories. As we followed this method, 230 indicators, 50 concepts, and five categories were derived. We also found unexpected trends emerged as we analyzed the data. For example, we found that early in interviews, practitioners made generalized statements about beneficial Agile practices. However, when they gave examples explaining how they dealt with challenges, they frequently described applying creative solutions to complex problems.

A useful technique during the interview process was use of probing questions to gain further insights. We used probing questions to increase confidence in our interpretation of incidents. For example, as Organization D gave an example of an inhibitor and we probed for influence on speed and/or stability. They responded by saying, "I'd say one of our biggest inhibitors to executing with speed was that while we were doing development, we were also setting up our entire base infrastructure. We didn't have a development region for the first four months…" This indicated to us that they perceived this inhibitor had an influence on speed. Had we had not probed we might have concluded that this problem influences stability only, rather than speed and stability. While we found the probing to be helpful, we also acknowledge that this approach may introduce a potential threat to validity. Investigator bias during probing could influence concept strength and, consequently, the ordering of practices.

Deriving concepts from 230 indicators taken from raw data transcripts can be overwhelming. For this reason we found the emergent categories very helpful in narrowing down the analysis scope and comparing "apples to apples". In this paper, we focused primarily on practices. However, several other

categories emerged including: Architectural Changes, Decisions, Constraints, and Deficiencies. In future work, we would like to investigate these other categories of enablers and inhibitors. We would also like to delve deeper into the interrelationships between all five categories and desired state. For example, we observed that constraints, such as slow management decision making, can inhibit rapid fielding, but what is their influence on achieving or maintaining desired state?

Because we were talking mostly with organizations using Agile development and Scrum, the majority of the examples followed the Speed-triggers-stability scenario. Another area we would like to explore is reversibility of the Speed-triggers-stability scenario. We see some evidence that the scenario is bi-directional. In other words, a focus on stability could trigger a focus on speed. For example, members of Organization E (one of the projects using an incremental lifecycle, but not using Scrum) explained they were losing market share because the development team was prioritizing work leading to an overemphasis on stability-related effort on the project. So, they "took the prioritization out of the hands of the developers" and created a product forum with involvement from business and technical sides. Clearly, emphasis on stability caused a refocus on speed. In future work, we would like to investigate whether examples of the stability-triggers-speed scenarios exist on Scrum projects and, if so, what are the similarities and differences between the response cycles.

V. CONCLUSIONS

Through this work, we see evidence that software engineers don't necessarily apply pure Agile or architecture practices separately. Practitioners we interviewed with come up with innovative ways to combine practices to allow them to stay within (or get back to) an acceptable range of desired state for their projects. We present several examples of this in this paper. These practice combinations allow teams to address problems with stability while still focusing on speed. We see value in investigating the practices that practitioners are combining with success to avoid commonly observed patterns of disruption to velocity.

We acknowledge a strong desire within the business community to avoid disruptive actions such as bug-fixing sprints or major redesigns. While the creative combinations of practices help to avoid major disruptions, the Speed-triggers-stability scenario is still a reactive pattern. This means that the problem has to surface before it is addressed. However, there may be value in investigating whether the combined practices could be applied in a more proactive way. For example, we see evidence from our interviews that improved visibility into technical debt may help avoid the disruptive big bang cycle. Are there other indicators that could be applied in a lightweight and dynamic way to position teams to be more proactive?

We identified several inhibitors, particularly, practice deficiencies, that pose great challenge to Agile projects. For example, testing appeared to be a particularly problematic area. The teams we interviewed struggled to develop test cases for complex situations and to run quality-attribute-focused tests, such as performance tests, within a sprint or release cycle. In addition, inconsistent test practices and slow assurance certifications processes led to problems and delays. This raises a question about whether we are observing a lack of testing practice discipline or a rational and reasonable response to the need to balance speed and stability.

Projects were able to respond to stability challenges in an incremental way when the problems were visible (e.g., painful to the business stakeholders). However, in the case of technical debt, where quality degrades due to shortcuts taken in the interest of speed, problems with stability were often not as visible to business stakeholders (e.g., tight coupling, which results in small changes taking a long time to implement). Due to the lack of technical debt measures, development teams could not make a strong case to the business side to convince them to invest in making the fixes. This often led to a disruptive bug-fixing sprint or major redesign. Because we could see an association between the lack of visibility into accumulating technical debt and disruptive measures, we suggest that this is an area for future investigation.

The Scrum Guide concludes with a statement suggesting that Scrum must be implemented in its entirety or the result is not Scrum [14]. Projects that tailor Scrum to incorporate their practices, or use portions of the Scrum method, are referred to in a derogatory sense as "Scrum But" projects. However, this sentiment appears to be changing. In a recent blog posting, Ken Schwaber said he would like to change the mindset of **Scrum But** to **Scrum And**. He explained that Scrum And is a path of continuous improvement in software development beyond the basic use of Scrum. He gave this example to illustrate the concept of extending Scrum, "We use Scrum **and** we are continuously building, testing and deploying our increments every Sprint" [16]. The work from this study supports the stance that practice extensions are needed and anticipated in iterative and incremental development. A recent study looking at practices across three Microsoft teams using Scrum also supports the notion that integrating engineering practices within the Scrum development framework improves software quality [18]. Future work exploring these integrated practices may reveal additional patterns for effectiveness, greater detail about how they work and shed light on aspects that may be generalizable.

ACKNOWLEDGMENTS

We thank the following individuals for their help and feedback during data collection: Deborah Brey, Christy Hermansen, Einar Landre and Josh Seckel.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This material has been approved for public release and unlimited distribution.

DM-0000071

REFERENCES

[1] M. Hotle, D. Norton, and N. Wilson, "The end of the waterfall as we know it," Gartner Research, August 2012.

[2] Director of Defense Research and Engineering, "Rapicapability fielding toolbox study," Final Report, March 2010. http://www.cogility.com/Documents/Rapid_Capability_Fielding-Public_Release.pdf

[3] M. Denne and J. Cleland-Huang, Software by Numbers. Upper Saddle River, NJ: Prentice Hall, 2003.

[4] F. Bachmann, R. L. Nord, and I. Ozkaya, "Architectural Tactics to support rapid and agile stability." CrossTalk: The Journal of Defense Software Engineering, Special Issue on Rapid and Agile Stability, May/June 2012.

[5] A. Martini, L. Pareto, and J. Bosch, "Enablers and inhibitors for speed with reuse," Proceedings of the 16th Software Product Line Conference, ACM, New York, v. 1, pp. 116-125, September 2012.

[6] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd ed. Boston, MA: Addison-Wesley, 2012

[7] T. Grant, "Navigate the Future of Agile and Lean." Forrester Research, January 2012.

[8] B. Dick, "Grounded theory: a thumbnail sketch," http://www.uq.net.au/action_research/arp/grounded.html (2005).

[9] G. Barney, "A look at grounded theory: 1984-1994," in Grounded Theory 1984-1994, ed., vol. I, Glaser, Barney G., Mill Valley, CA: Sociology Press, 1995, pp. 3-17.

[10] J. Corbin and A. Strauss, Basics of Qualitative Research-Techniques and Procedures for Developing Grounded Theory, 3rd ed. Thousand Oaks, CA: Sage Publications, 2008.

[11] S. Adolf, W. Hall, and P. Kruchten, "A methodological leg to stand on: using grounded theory to study the experience of software development," Vancouver, BC: University of British Columbia, January 2011.

[12] D. Leffingwell, Scaling Software Agility. Upper Saddle River, NJ: Addison-Wesley, 2007.

[13] B. Glaser, The Grounded Theory Perspective: Conceptualization Contrasted with Description. Mill Valley, CA: Sociology Press, 2001.

[14] K. Schwaber and J. Sutherland, "Scrum guidebook," Scrum.org and Scrum Inc., 2011.

[15] Keith H. Bennett and T. Rajlich, "Software maintenance and evolution: a roadmap," in Proceedings of the Conference on The Future of Software Engineering (ICSE 2000). New York: ACM, 2000, pp. 73-87.

[16] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: from metaphor to theory and practice," IEEE Software, 2012, pp.18-21.

[17] K. Schwaber, (blog) "Telling it like it is," April 2012. http://kenschwaber.wordpress.com/2012/04/05/scrum-but replaced-by-scrum-and/

[18] L.Williams, G. Brown, A. Melzer, N. Naggappan, "Scrum + Engineering Practices: Experiences of Three Microsoft Teams", ICSE 2013.