12-17-2014

The Vortex of Continuous Development of Embedded Systems: An Inquiry into Agility Orchestration

David A. Bishop
Georgia State University

PERMISSION TO BORROW

In presenting this dissertation as a partial fulfillment of the requirements for an advanced degree from Georgia State University, I agree that the Library of the University shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to quote from, to copy from, or publish this dissertation may be granted by the author or, in his/her absence, the professor under whose direction it was written or, in his absence, by the Dean of the Robinson College of Business. Such quoting, copying, or publishing must be solely for the scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of this dissertation which involves potential gain will not be allowed without written permission of the author.

David Anthony Bishop

NOTICE TO BORROWERS

All dissertations deposited in the Georgia State University Library must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this dissertation is:

David Bishop
271 River Laurel Way
Woodstock, Georgia 30188

The director of this dissertation is:

Richard Baskerville
Robinson College of Business
Georgia State University
35 Broad Street, NW, Suite 427
Atlanta, GA 30303

The Vortex of Continuous Development of Embedded Systems: An Inquiry into Agility Orchestration

BY

David Anthony Bishop

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree

Of

Doctor of Philosophy

In the Robinson College of Business

Of

Georgia State University

ACCEPTANCE

This dissertation was prepared under the direction of the David Bishop Dissertation Committee. It has been approved and accepted by all members of that committee, and it has been accepted in partial fulfillment of the requirements for the degree of Doctoral of Philosophy in Business Administration in the J. Mack Robinson College of Business of Georgia State University.

H. Fenwick Huss, Dean

DISSERTATION COMMITTEE

Richard Baskerville
Lars Mathiassen
Balasubramaniam Ramesh

Table of Contents

List of Tables

List of Figures

Abstract

The Vortex of Continuous Development of Embedded Systems: An Inquiry into Agility Orchestration

BY

David Anthony Bishop

May 6, 2014

Committee Chair: Richard Baskerville

Major Academic Unit: Robinson College of Business

Agile methodologies have become a popular and widely accepted method for managing software development. Since the inception of the *Agile Manifesto* over ten years ago, agile development techniques have superseded waterfall methods in many, if not most, software development organizations. Despite its apparent success, many companies have struggled with the adoption and implementation of agile, and exactly what level of adoption provides optimum agility. Agility is commonly held in the literature to be constructed of elements external to a company or project but may in fact be composed of both external and internal elements. The exact relationship of the *adoption* of agile development techniques and their relationship to the *actual agility* of a business remain unclear. A primary contributor to this uncertainty is the somewhat amorphous definition of agile itself. In academic literature, the concept is still relatively young and loosely defined. In practice, organizations have largely opted for a hybrid approach to agile, mixing its concepts and methods with existing Stage Gate or waterfall methodologies. This has made the management of agile even more complex. Crucially, there is no definition or criterion available to determine the *appropriate mix* of agile and waterfall processes in an embedded software development context nor is there a method to determine the

impact of one against the other. These issues beg the question: how do organizations manage agility? This interpretive case study provides an empirical account of how stakeholders manage both market and process agility in an embedded systems context via a hybrid agility implementation and product genesis. As a result, we provide the notion of *agile vorticity*, as the point at which market and process agility collide to produce *business momentum* at a specific point of innovation within the *agile business vortex*.

Chapter 1: Introduction

Research Domain

The management of agile methodologies can be a challenge to organizations on many levels. The first issue is the concept of agile itself. The current literature on agility is sparse, particularly in relation to information systems development. This is largely due to the fact that agile is a relatively new concept in an information systems development context, and is therefore not entirely solidified. The second challenge is the management of agility. Most agile frameworks available today are operational in nature, focusing on project management indicators such as velocity, release frequency, sprint completion, and so forth (Highsmith, 2010). Still, other methods focus entirely on the outcome of agile adoption in relation to environmental turbulence (Yauch, 2011). This type of study, however, does not allow for the evaluation of the adoption of agile principles, such as people over processes and tools, against quality and customer responsiveness. Giachetti holds that the assessment and management of agile should not only take into account performance, but agile characteristics that have been assimilated into the organization (Giachetti, Martinez, Sáenz, & Chen, 2003). Although there have been methods describing agile adoption, none of these has related the level of adoption to agility outcomes. As a result, these existing methods do not completely address all dimensions of agility. These challenges are compounded by the fact that recent research has revealed that most organizations have not adopted agile in its entirety, but instead have assimilated a variety of agile concepts and methods into existing traditional methods. Such *hybrid agile* implementations have only added to the complexity of agility management.

Orchestration of agility within any organization is often poorly defined, due to its fluid, quickly changing, and somewhat amorphous concepts, processes, and methodologies. Hybrid

implementations, environment specific assessments, and varying degrees of market turbulence are just a few dimensions of agility that should be considered within a management context. In short, the management of agile methods require their own brand of agility. This study examines the various dimensions of this agility to determine how agile processes are managed and orchestrated.

Research Perspective

Transitioning to, adopting, and implementing agile methodologies in a software organization is a costly and time consuming proposition. Companies and organizations need to know where they stand in terms of adoption/assimilation and its impact to the external agility of the business. They also need a framework to manage their performance so that strategic adjustments can be made. For researchers, this study may provide greater understanding with regards to how agility can be managed and the impact of assimilation and adoption on these management processes. As the literature review has revealed, most embedded systems environments with larger, more mature organizations have taken a hybridized approach to agile adoption by combining agile with existing State Gate methodologies. The orchestration of the processes required to support agile principles in such an environment and the way those principles are affected can reveal new insights into agility in new and different contexts. Such contexts include

- large teams,
- geographically distributed teams,
- hybrid agility evolution, and
- embedded systems (environments which include the synchronization of firmware, software and hardware development).

From a practitioner standpoint, the audience for this study includes technology managers, software development professionals, and organizations interested in adopting agile methodologies or managing agile processes in their current environment. From an academic perspective, this study would appeal to researchers interested in the adoption, implementation, or assimilation of agile methodologies in different organizational contexts.

The purpose of this study is to provide an empirical account of the orchestration of hybrid agility, and feedback on how this orchestration is adjusted to integrate agile principles more fully into a complex, embedded systems development environment.

Therefore, to better understand agility and how organizations adapt to it, this research endeavors to answer the question: How are agile processes orchestrated in embedded systems development?

Research Approach

This research consists of an interpretive case study using a grounded theory analysis. The grounded theory approach makes use of the Straussian brand of grounded theory outlined in Strauss and Corbin's *Basics of Qualitative Research: Grounded Theory Procedures and Techniques* (Strauss, 1990). Such studies can provide theories arising from the research effort itself, and therefore do not generally employ theoretical frameworks as a lens to examine a problem. However, such studies can use other research to inform the study at hand (Strauss, 1990). This case study uses the four basic agile principles as stated in the *Agile Manifesto* for this purpose. These principles can be described as

- development over documentation,
- individual interactions over processes and tools,
- customer Collaboration over contract negotiation, and

- responding to change over following a plan (Alliance, 2001).

This is a participative study that includes perspectives on agile from a variety of stakeholders within the company in question. It examines the normative questions dealing with the management, design, and evaluation of agile assimilation specific to embedded systems development. These traits are characteristic of a engaged scholarship effort using design and evaluation research (Van de Ven, 2007).

Contributions of the study include

- an empirical account of how agility principles and methods are orchestrated in embedded software development, and hence, the orchestration of hybrid agility,
- in keeping with engaged scholarship principles, feedback is provided to the client organization on how its approach to agility could be improved or "tweaked" as a component of software development management,
- a method or framework for orchestrating agility over time in context, while constantly adapting and fine tuning it, and
- greater insights into the drivers of agile process innovation.

We begin with an exploration of engineering and software traditions, followed by agility, embedded systems, and the current state of research on each.

Chapter 2: Approach: Agile Software Development Methods

Engineering and Software Traditions

In the past, there has been much debate as to whether software development is science or engineering. In the literature, most researchers believe it to be the latter. This perception is due in large part to the difference between the tools that scientists and engineers use. While a scientist may use experimentation, the engineer develops prototypes or demonstrations. In his study on the traditions of computing, Tedre stated that "computer science is an empirical science not based on traditional scientific experimentation" (Tedre, 2008). Over time, software development became more regarded as an engineering discipline as the size and scale of computing projects required larger teams (Tedre, 2008). As a result, the guiding mantra for software engineers and computer scientists is more often considered to be *demo or die*, as opposed to the traditional *publish or perish*, employed by so many traditional researchers. These precepts have had a significant influence on the management of large software development efforts. This section seeks to explore the evolution of software development management and how this evolution has been influenced by engineering, science, and innovation.

Although larger teams and more complex development projects have led to software development becoming more of an engineering discipline, this has also led to a need for a more efficient means of management. Early software engineering managers and developers were faced with challenges in design, development, communication, and production. Arguably, the earliest form of software development management is traditionally referred to as the software development life cycle or *waterfall* method. It is also often referred to as a Stage Gate method due to its rigid process orientation and the use of *gates* to pass from one phase to another. The creation of this method is most often attributed to Winston Royce in 1970 when he documented

the process he used to develop software packages for spacecraft mission planning (Royce, 1970). Although not originally intended to be a comprehensive roadmap, Royce's work proliferated throughout other government agencies and became widely adopted within the software development industry. According to Christiane Floyd in her 1992 publication of "Software Development and Reality Construction" this software development tradition is based on the following precepts (Floyd, 1992):

- Software engineering is produced, based on a series of fixed requirements.
- These requirements are provided via an analysis process that is performed before design begins.
- Developers are only responsible for producing a solution that meets these specific requirements.
- Software development is independent of individuals. Developers are considered to be interchangeable resources.
- Communication should be managed and regulated through fixed interfaces.

Floyd argues that while this existing methodology has brought about impressive advances in programming methods and allows for a greater understanding of software development before coding begins, it does not support the subsequent emergence of insights into functionality, implementation, and usability (Floyd, 1992). The need to provide such insights gave rise to a greater usage of the engineering concept of prototyping.

The concept of prototyping, or even rapid prototyping, is not new. Rather, it is a tradition that has been a part of electrical and mechanical engineering for many years. One reason for its success is that it allows for more teleological requirement definitions based on outputs and constraints (Orr, 2004). Back in 1985, many authors believed that prototyping would replace

traditional life-cycle methods of systems development (Janson, 1985). However, a study from this time period does not recommend that. Instead, it performed a comparison of prototyping based on existing engineering practices against proposals to use the same technique in information systems design. This comparison suggested that prototyping should be integrated within existing waterfall processes to

- verify user requirements,
- verify design specifications,
- aid in selecting the best design,
- assist with various stages of testing and development, and
- obtain approval for new product concepts (Janson, 1985).

This need for prototyping in software has given rise to iterative software development in which regular demonstrations of incremental components are key to developing an entire product line.

Manufacturing concepts have also had a significant impact on software development. The concept of lean or *just in time* manufacturing became prominent in the 1970's as part of the highly regarded Toyota production system. Lean development is based on removing any aspect of a process that does not add customer value. At Toyota factories, inventory was kept to a minimum by only manufacturing the necessary products, at the necessary time, in the necessary quantities. In addition, the system included a *respect for human* system which allowed workers the latitude to display their capabilities by improving their own work processes (Sugimori, Kusunoki, Cho, & Uchikawa, 1977). Lean techniques improved the flow of information and materials across the business, focused on market pressure created by the customer, and required an organizational commitment to continuous improvement. Although lean manufacturing was

driven in large part by Japan's need to compete with Europe and America with fewer resources, the concepts became widely adopted in the automotive industry all over the world.

The production control system of the Toyota production system was referred to as the *Kanban* system, which consisted of a series of order cards instead of computers to manage production (Sugimori et al., 1977). Kanban is yet another outgrowth of the Toyota production system which has made its way into software development management. Using Kanban, software development organizations have been found to reduce the amount of work in progress. Such lean concepts of value and waste elimination have been found to provide *target and route* for continuous software development improvement, particularly with agile development (Wang, Conboy, & Cawley, 2012). Organizations that employ Kanban techniques have moved away from the time-boxed iteration to more of a *continuous flow*. This has been shown to be especially true in organizations with mature adoption of agile development techniques (Wang, Conboy, & Cawley, 2012).

Agility

Agility originated in a manufacturing context, primarily as an output of lean or flexible manufacturing (Mathiassen & Pries-Heje, 2006; Kidd & Dove). It concerns the economy of scope, as opposed to scale (Dove, 2001). It has been defined as the ability to manage and apply knowledge effectively, to adapt to change (Arteta & Giachetti, 2004; Dove, 2001), and it has been summarized as the capability to quickly respond to market requirements (Ramesh and Devadasan, 2007).

The concept of agility was first introduced in a report from the Iacocca Institute (Nagel and Dove, 1992). Other notable research articles include:

- A Hewlett-Packard (HP) agility assessment expands on agility as comprising three factors of speed, range, and ease to assess an organization's ability to respond to change. (HP 2005).
- Agility was defined by Goldman et al. (1995) as the ability to prosper in a competitive environment characterized by constant and unpredictable change. This concept was further broken down into four dimensions of agility, which are
    - enriching the customer,
    - cooperating to increase competitiveness,
    - organizing to control change and uncertainty, and
    - leveraging the impact of people and information.

Haeckel expands on these dimensions in a different way, by enumerating the organizational characteristics required to achieve agility as described in Goldman's dimensions. Speed with which an organization can respond to customer requests, market dynamics, and emerging technical change is seen as a key element. This includes

- time to sense relevant events,
- time to interpret what is happening and assess the consequences for the organization,
- time to explore options and decide which actions to take, and
- time to implement the appropriate responses (Haeckel 1999).

Organizational capabilities, both tangible and intangible, that provide the basis for conducting business and creating change are also considered to be a prerequisite to achieve agility. These include people, technology, processes, and knowledge (Haeckel 1999).

Adaptability is essential as well. How well organizations respond to changing demands, threats, or opportunities require the ability to learn and to use flexible processes and products that can be reconfigured without extensive additional costs (Haeckel, Dove; Mathiassen & Pries-Heje, 2006).

Agility has perhaps been best described as a solution for maintaining competitive advantage during times of uncertainty and turbulence in the business environment (Sharifi & Zhang, 2001).

Further research into the literature reveals that agility is more than a method or an organizational capability. Rather, it is a business philosophy (Highsmith, 2010). As agile concepts champion people over processes, focus should be on the organizations and the individuals that comprise them. Agile *minds* should be quick, resourceful, and adaptable in character. Agile *organizations* should respond quickly, be resourceful, and able to adapt to their environment (Mathiassen & Pries-Heje, 2006).

Organizations are complex adaptive systems. Such systems have been defined as comprising of decentralized independent individuals interacting in self-organizing ways, guided by a set of simple, generative rules, to create innovative emergent results (Highsmith & Cockburn, 2001). Highsmith and Cockburn emphasize creativity over written rules as the way to manage complex software development problems and diverse situations. (Highsmith & Cockburn, 2001). It is this style of management from which *organizational* agility is said to arise. Additionally, organizations do not think objectively about software development agility (Sheffield & Lemétayer, 2013). Adoptions of agile practices are typically subject to the organizational structure and context. In one study, it was found that developmental organizations, those which focus on adaptation and creativity, were much more conducive to the

adoption of agile practices than a hierarchical culture which focused more on command and control (Iivari & Iivari, 2011).

Although the concept of agility has been available in manufacturing for some time, it was not adapted to information systems development until the advent of the *Agile Manifesto*. Information systems have brought about a new context and application for agility. An information systems development's ability to support agility has been defined as the continual readiness to rapidly or inherently create change, embrace change, and learn from such change while contributing to perceived customer value. Such value is often characterized as additions to economy, quality, and simplicity. This is accomplished via the information systems development's collective components and its relationships with its environment (Baskerville, Pries-Heje, & Madsen, 2011).

Based on the literature that has been discussed, it can be noted that agile is somewhat conceptually weak from a research point of view due to the number and variation of definitions. In practice, however, it is considered well defined enough for *practitioners,* particularly with respect to how they use and combine agile with plan driven methods (Baskerville et al., 2011).

Agile Software Development Methodologies

Agile is an iterative software development methodology based on self-organizing and cross-functional teams. It is based on the following key concepts derived from the highly popularized *Agile Manifesto* (Alliance, 2001; Vinekar, Slinkman, & Nerur, 2006) that argues

- individuals and their interactions are more important than processes and tools,
- working software is more important than documentation,
- customer collaboration is more important than contract negotiation, and
- responding to change is more important than following a plan.

Using Ward's method, Dingsøyr al. identify the seminal works in agile research and many of the key underlying themes. Most of the early research focused on understanding agile concepts. Other key topics included adoption or adaptation, reconciliation between agile and plan-driven methods, and evaluation of adoption issues in environments not conducive to agile (Dingsøyr, Nerur, Balijepally, & Moe, 2012). More research is needed to better define what the core of agile is and its role in architecture and knowledge management (Dingsøyr et al., 2012). Additional research is also needed with respect to examining agile across various contexts such as different projects and organizations.

One such context that demands study are methods for information systems development. An information systems development method can be defined as one that "encompasses the complete range of practices involved in the process of designing, building, implementing, and maintaining an information system, how these activities are accomplished and managed, the sequence and frequency of these activities, as well as the values and goals of all of the above" (Conboy, 2009). Such a method is not a set of rules, but an ideal in the sense that it is not expected to be followed literally (Conboy, 2009).

Conboy finds that an agile information systems development method should meet the criteria of

- flexibility—the ability to create change, or proactively, reactively, or inherently embrace change in a timely manner, through its internal components and relationships with its environment;
- leanness—the ability to contribute to perceived customer value through economy, quality, and simplicity from the *customer's perspective*.
- Agility— the combination of flexibility and leanness with continual readiness.

Qumer adds speed, learning, and responsiveness to these criteria (Qumer & Henderson-Sellers, 2008). One problem with current agile method thinking is that some practices are now commonly referred to as agile even though the connection to the concept may be tenuous at best, and even if this link is clear, it may be too simplistic to be considered agile in every context or circumstance (Conboy, 2009). Conboy states that the following steps should be taken to evaluate such practices based on the aforementioned criteria:

1. Evaluate whether certain practices or procedures are agile with respect to long-term sustainability and implementation.
2. Examine the *behaviors* and *outcomes* that contribute to agility.

This idea could be extended by developing assessments to evaluate performance outcomes (Conboy, 2009). Applying such assessments across methods, method variants, organizations, and projects could not only reveal interesting insights, but improve orchestration of agile processes.

The integration of agility concepts into information systems development has created a number of variations on a theme. These include adaptive systems development, dynamic systems development, test driven development, and feature driven development to name a few. By far the most popular of these methods used in the industry today are XP and Scrum (Baskerville et al., 2011). A more recent addition to this list of commonly used methodologies is Kanban.

Scrum is more of an agile management methodology with a greater focus on projects, while XP is more of an engineering philosophy, concentrated on code management and quality (Wang, Conboy, & Cawley, 2012). Both methods can, and often are, used in tandem. Weaknesses with XP have been cited with medium to large size projects because of inadequate

testing, architectural planning, and documentation. All of which are often required with large complex systems in which the cost of change can be high (Qureshi, 2012). Studies have shown that some shortcomings such as defect rates can be mitigated by extending XP to include more analysis and architectural design, characteristics which look similar to waterfall-based methods (Qureshi, 2012). For the purposes of this study, we are primarily focused on the management aspects of agile and therefore the focus will be on Scrum. Another reason for this emphasis is that Scrum is commonly used in hybrid implementations of agile methods. This is mostly because Scrum acts as a wrapper around existing development methodologies, and can be used with virtually any existing method (Schatz & Abdelshafi, 2005).

Despite its success, Scrum has been found to present challenges with resource allocation. This has led to a noticeable shift in the industry from agile to lean software development practices (Wang, Conboy, & Cawley, 2012).Kanban is one such process, and it is a less structured method than Scrum, which focuses on minimizing the amount of work in progress. Instead of using time-boxed iterations, Kanban employs more of a flow by allocating time and resources as they are needed. Software development organizations that have challenges with work estimation and interruptions have been shown to show improvements over lead time and defect rates by using Kanban over the time-boxed method of Scrum (Sjøberg, Johnsen, & Solberg, 2012). A significant area for further research is operational guidance on mapping such lean and agile processes to their current roles and a roadmap for implementing them (Wang, Conboy, & Cawley, 2012).

Successful implementation of an agile methodology has been found to rely heavily on the establishment of many cultural and procedural changes within an organization. The first of these is building a continuous feedback loop to allow for constant replanning (Vidgen, 2009). Shared

responsibility by empowering Scrum team members to manage day-to-day work is also important (Vidgen, 2009). A key enabler for self-managed teams is fostering high communication and collaboration on a daily basis. Spontaneous interactions should be supported by structured interconnected practices such as Scrum meetings and pair programming (Vidgen, 2009). A willingness to adapt the process to the development context is key, and the development iterations themselves should work towards a sustainable rhythm (Vidgen, 2009).

Agile implementation is not limited to the development organization. Product management should also integrate agile methods into their work. This can be accomplished by establishing sprints that alternate with the development teams. Implementing agile in product management as well as development provides for structured detailing of complex requirements, early collaboration, and disciplined backlog administration. (Vlaanderen, Jansen, Brinkkemper, & Jaspers, 2011)

For requirements prioritization, it was found that a mix of agile and plan based methods proved to outperform either agile or plan based methods alone in which volatility is not very high or low (Port & Bui, 2009). Since volatility is rarely at the extreme and often unknown, it was inferred that mixed strategies should be the most widely used. However, it should be noted that very turbulent markets or *gold rush* situations could accelerate the volatility or "pull" rapidly. The adoption of such mixed practices have been found to allow for change, driven by close customer interaction, continuous requirements gathering, and frequent iterative delivery (Vidgen, 2009).

Although agile has been noted to increase productivity, foster shared learning, and create job satisfaction among developers (Vinekar et al., 2006), it may not be the best choice for all

environments. The literature as well as practitioner experience shows that successful adoption of agile in its purest form may depend on the following factors:

- size of the project and team,
- consequences of failure or criticality of the project,
- volatility of the environment,
- skill level of the development team(s), and
- company culture (Vinekar et al., 2006).

One example of such an environment, embedded systems development, is impacted by these factors on many levels. Firmware and hardware development teams tend to be much smaller than their software counterparts with a much higher degree of specialization. While there are many software developers in the organization with skills that are easily transferable from one project to another (such as C#, Java, or .NET programming), their firmware counterparts do not share the same level of transferability. Firmware professionals often have very focused knowledge of the embedded systems stacks for home area networking, RF network communication, or metering metrology that inhibits them from being easily interchangeable. Embedded systems are often *mission critical* systems with high consequences for failure. As a result, organizations developing such systems tend to be less comfortable with the higher rates of change that often come with the iterative and potentially chaotic agile development than their software counterparts.

Chapter 3: Context: Continuous Development of Embedded Systems

Embedded Systems

An embedded system is typically one which consists of a combination of software, firmware, and hardware components that must be developed and tested in tandem (Douglass, 2004). The exact nature of an embedded system may vary according to the application, but it typically consists of the following key characteristics:

Embedded Systems are Real Time

A real time embedded system is one where the predictability and schedulability of the system affects the correctness of the application (Stankovic, 1996). For example, if a purchase is made on a website, it may appear as though the transaction is immediate or *real time*, but in fact, it is being queued on a server and is being processed accordingly. If it takes a few extra minutes to process a credit card, the functionality of the application is not affected. However, in the context of a complex avionics system on a fighter jet, or a heart pacemaker, if a signal is not sent correctly at a specific instance in time, disastrous consequences could result, meaning that the application has failed. This also applies to smart metering devices. If meter readings are not calculated and sent at a specific instance in time, incorrect billing could result. If the meter fails to respond to a load shedding event or power interruption, this could cause problems on the electric power grid.

More stringent reliability and safety requirements are needed for embedded systems. Such requirements call for extensive fault tolerance and safety testing to ensure that the equipment in question is safe for workers to manage and that the public interest in protected (Stankovic, 1996). In the case of smart metering, much of the equipment may have thousands of volts flowing through it at any given time. Safety guidelines must be determined through

extensive testing. Utilities that purchase this equipment must answer to the public service commissions of their respective governments, which often have questions regarding accuracy and reliability.

Complex Change Management

Most IT systems are maintained systems. In other words, the software work they entail consists of small incremental efforts to add features and repair defects. This work is easily conducive to an iterative approach. By contrast, an embedded system contains software, hardware, and firmware that are intertwined. Making a change to an embedded device, particularly once a few thousand circuit boards have already rolled off the assembly line, could be a monumental undertaking both technically and financially.

More device driver-level software is required—as implicated in earlier statements, embedded systems often have custom hardware, requiring custom software drivers to operate them (Douglass, 2004). A smart meter is a good example. It consists of a radio, metrology, and home area network hardware components, all of which are operated by custom software applications. If one of these components changes, others may be impacted as well.

More restrictive optimization requirements are also needed due to the highly resource-constrained platforms (Douglass, 2004). Most embedded devices only have a very limited amount of memory and CPU to operate on.

Finally, there is a significant difference in target environments between embedded systems and traditional software applications (Douglass, 2004). A typical software application is developed on a PC and can be installed, tested, and run in the same or very similar environment. An embedded system however, must be developed on a different environment from its target. For example, a smart metering application may be developed on a laptop computer, but it must

be tested and operated on a smart meter or radio frequency network equipment. The characteristics of these environments are difficult to simulate, and so they must be tested as much as possible with real equipment. Due to size and cost restrictions, full scale testing can be exponentially more complicated than traditional software applications.

All of the aforementioned characteristics combine to significantly differentiate embedded systems from the traditional software application development effort. In line with these characteristics, agile is often not considered for embedded systems development, due to lack of full life cycle support and tools (Smith, Miller, Huang, & Tran, 2009).

Despite these barriers, embedded systems development organizations have successfully integrated XP and Scrum based agile practices with positive results. One such successful example provided for acceptance testing that drove a high level of prototyping, beyond what standard XP development would provide (Smith et al., 2009). As one may expect from such examples, the literature shows that the integration of specific practices and their adaptation varies from company to company and from project to project (Salo & Abrahamsson, 2008; Sue, Kendall, & Kendall, 2012). Most critically, a study of process model selection in embedded systems development found that for large, complex projects no single method applied most of the time, rather, a "hybrid model blending and balancing the features of different models is often the choice" (Kettunen & Laanti, 2005).

Continuous Development

The concept of continuous software releases consists of providing for a series of smaller, sequential releases, as opposed to one large monolithic production. Such concepts have been long known to provide better time-to-delivery for software products. More specifically, continuous development has been found to have the following benefits (Greer & Ruhe, 2004):

- Requirements can be prioritized so that a working, beneficial system can be produced sooner.
- It allows customers to receive at least a piece of a working system earlier and provide feedback.
- It allows for the integration of customer or user feedback at incremental stages.
- It simplifies scheduling and estimation due to working with *smaller chunks* instead of larger products.
- It makes adapting to change easier.

The key components of continuous development have been found in the literature to be release planning, iteration pacing, and continuous integration of change.

Release planning or road mapping is important to directing iterative releases. In one study, a systematic review of 24 release planning models was performed, and sixteen of these belonged to the EVOLVE family of models. EVOLVE employs a genetic algorithm to determine an optimal requirements set for each iteration (Greer & Ruhe, 2004). It can be used to build a release plan within certain technical constraints once requirements have been categorized and estimated. Most planning methods found in the literature focus on a small set of requirements selection factors and emphasized constraints such as budget, technology, and schedule. About 58% of these methods included soft factors such as customer or company value, risk, stakeholder

influence, or resources. Although most of these models were validated with case studies, very few were tested in full-scale industry projects. Additionally, all such models were intended for market-driven development (Svahnberg et al., 2010). In essence, Svahnberg's study revealed that there are few real choices for practitioners wishing to adapt a release planning model, and the most of those in existence are very interrelated. Finding a model that suits a company's unique needs, which at the same time has enough empirical evidence from industry to prove that it works, is challenging (Svahnberg et al., 2010).

Continuous releases typically require the creation of successive development iterations. Iterations can be used from the product development level down to the organization of individual coding tasks. Companies that compete in very fast-paced markets have found that proactively setting a time-boxed pace for new product development, based on an established *rhythm*, allows them to keep one step ahead of the competition (Eisenhardt & Brown, 1998). It does this by combining flexibility and control in turbulent environments (Vidgen, 2009). This is in contrast to *event pacing* which is more reactionary. Although every market will have its surprises that require companies to be reactionary at some point, making proactive commitments to innovation in this way has been shown to have a direct impact to the timeliness and effectiveness of new product introduction (Eisenhardt & Brown, 1998). A key in making time pacing successful is the use of time based performance metrics such as speed, rate, and elapsed time, in addition to costs or profit margins.

Another component of continuous development is the practice of continuous integration of development changes. Continuous integration has been found to increase quality at up to 30% because it eliminates the integration periods required by the delivery milestones of a traditional systems development life cycle. (Karlstrom & Runeson, 2005; Schatz & Abdelshafi, 2005).

Although often considered an important contributor to the success of extreme programming methods, a recent study concluded that the concept is "not homogeneous and has many contextual variations" (Ståhl & Bosch, 2014). The study identified a need for a model that described these variants and their effects. Industry stakeholders could then decide which variant they should seek out based on their respective goals. As with time pacing, more advanced tools or reporting systems are needed to allow for greater user input into this continuous integration flow (Muthitacharoen & Saeed, 2009).

Chapter 4: Hybridity: Adoption of Agility to Context

Adoption of Agility: A Hybrid Approach

The push to adopt agile development methodologies within the technology sector is strong. Many companies have been made to feel that adoption of agile is critical to staying competitive. This has become so pervasive that even the government has taken notice. In a recent study, the Department of Defense cited insufficient progress and performance with traditional methods, and inability to provide urgent responses to evolving mission needs, as key reasons for adopting agile methods (Broadus, 2013).

Despite the drive to adopt, assimilation of agile methods into new organizations well entrenched in traditional waterfall methodology often face significant resistance. Many stakeholders are uncomfortable with key tenets of the *Agile Manifesto* and fear that loosely defined requirements and iterative development will cause significant disruption, particularly in complex projects (Barlow et al., 2011).

Organizational, process, and procedural barriers to agile adoption are numerous. Some of the most commonly cited concerns are the management of non-functional requirements, documentation, contractual issues, resource management, and cost estimation (Barry Boehm & Turner, 2005). Depending on the industry, conflicts with critical design review processes, regulatory requirements, and human resource policies can also be difficult to overcome. Additionally, maturity assessments and traditional engineering performance indicators can become issues. (Boehm & Turner, 2005) Critics often doubt whether the benefits of agile outweigh the costs of adoption. Many cite the lack of required documentation and too much focus on coding, as opposed to implementation and planning. Others have noted implementation failures in large complex projects (Barlow et al., 2011). In some agile environments

development was found to skip procedures or processes under tight deadlines, causing the process to proceed in an ad-hoc shortsighted way (Karlstrom & Runeson, 2005). Such concerns tend to be more pervasive with embedded systems development.

Some studies found, not long after the advent of the *Agile Manifesto*, that although agile methods were designed to solve many of the same problems that faced embedded development, existing methods were not well suited to the task (Ronkainen & Abrahamsson, 2003). Embedded systems often place the following constraints on agile assimilation (Ronkainen & Abrahamsson, 2003):

- Up-front architecture design cannot be avoided and must be provided for.
- Refactoring must include configuration management for both software and hardware, supported by system level analysis.
- Transitioning prototypes to well documented production code requires techniques for increasing code maturity.
- More formalized communication and coordination methods are needed between teams.
- A method is needed for throttling changing requirements gradually as the product gets closer to release.
- Techniques are needed for building and optimizing test cases.

Integration of agile with Stage Gate methodologies addresses many of these constraints such as resolution of communication problems (Karlstrom & Runeson, 2005) . Such dependencies are even more prevalent in an embedded systems environment. In a study of agile integration with software product line engineering, it was found that although collaboration between teams is encouraged, project managers should also manage the boundaries between

teams. Additionally, project managers should manage the scope based on market, organizational, and technological factors (Mohan, Ramesh, & Sugumaran, 2010). Software product line engineering is similar in complexity to embedded systems development, in that it involves the development of one comprehensive solution that may span multiple domains.

An organization need not be concerned with complete adoption of all commonly accepted agile practices. Environments dictate practices as opposed to principles, and some combinations of practices may be better suited for specialized environments (Baskerville, Ramesh, Levina, Pries-Heje, & Slaughter, 2003). Some of the latest research has shown that many adopters, particularly larger organizations, are taking a hybrid approach, stating that a la carte selection of agile practices can work very well. (Fitzgerald, Hartnett, & Conboy, 2006). Other studies have shown that combining agile with other approaches has proved promising and that practitioners should not be afraid to adopt hybrid methods tailored to their needs (Sheffield & Lemétayer, 2013). In fact, a one size fits all solution for software agility has been found to be inappropriate because it is often contextual and organizationally dependent (Sheffield & Lemétayer, 2013; Sue et al., 2012). Mixed strategies for development work such as requirements prioritization have been found to outperform either agile or plan-based methods alone (Abrahamsson, Conboy, & Xiaofeng, 2009). Hybrid approaches are further strengthened by the fact that they build on the strengths of both plan-based and agile methods while mitigating their weaknesses (Barlow et al., 2011). For example, agile methods have been shown to provide the Stage Gate model with tools for planning small iterations, day to day work management, and reporting. In turn, the Stage Gate model can provide agile methods a means to coordinate with other development teams and communicate with marketing and upper management (Karlstrom & Runeson, 2005). Most critically, much research has shown that agile is often not the best choice in certain contexts,

such as larger organizations, and combining it with State Gate models has proven the best approach. (Dybå & Dingsøyr, 2008). Such contexts need development approaches that balance flexibility and disciplined methodology (Baskerville et al., 2003). The causal factors for such a hybrid approach have been stated to include

- a desperation to rush to market,
- a new and unique software market environment, and
- a lack of experience of developing software under the conditions imposed by the environment (Baskerville et al., 2003; Lyytinen & Rose, 2005).

Indeed, those facing high uncertainty and reciprocal interdependencies in their projects should implement a hybrid method combining strengths of current software life cycle development with complementary agile practices (Barlow et al., 2011).

Finally, this trend of hybridization is expected to continue and proliferate, perhaps to the extent of changing the face of agile development methods themselves. The proliferation and assimilation of agile development methods is a cyclical evolution that continues to this day and will ultimately combine agile and plan driven techniques (Baskerville et al., 2011).

As mentioned previously, agile is as much a business philosophy as it is software development methodology. With its adoption comes an entire change in the way a company does business. Design reviews are handled differently, product delivery is iterative, and the management of expectations relating to acceptance and decision making have to change (Broadus, 2013). In fact, adopting agile often requires moving away from the iron triangle of cost, scope, and schedule into an entirely new project management paradigm (Baskerville et al., 2003; Highsmith, 2010).

Successful agile adoption depends not only on agile development teams but agile organizations. As mentioned earlier, product management and executives must also participate in the process (Vlaanderen et al., 2011). Particularly in large or relatively mature companies, it is necessary to focus on human and social interaction to succeed (Dybå & Dingsøyr, 2008). In such organizations, the high levels of *individual* autonomy provided by agile must be balanced with high *team* autonomy and corporate responsibility (Dybå & Dingsøyr, 2008). This requires teams with high functioning employees capable of trust, strong communication skills, interpersonal skills, and confidence in their own abilities (Dybå & Dingsøyr, 2008). Complete integration of agile at the organizational level requires an understanding of the adopters, understanding of the risks, and time to allow change to work (Broadus, 2013).

Environments can dictate practices, and studies have shown that some combinations work better in different environments than others based on project size and other factors (Baskerville et al., 2003). Barlow provides a framework to evaluate what kind of agile approach is best for a given situation based on the examination of project interdependencies and volatility. It was found that large mature organizations often require a hybrid approach to agile due to complexity, IT governance processes, and size of the teams (Barlow et al., 2011). Additional studies made similar conclusions. Using adaptive structuration as a lens, one study concluded the following points regarding optimum agile adoption (Baskerville et al., 2003):

- Successful adoption requires top management buy-in and support.
- Methods should be tailored to the team.
- Developers need to understand the impact of their autonomy.

Key to making all three of the points above happen is communication. Indeed, an organization's support for formal and informal communication affects the outcome of the type of hybrid approach used (Barlow et al., 2011).

Despite the extensive monetary and organizational commitment required to adopt agile practices, it is rare to have any comparable data to explain the impact of agile before and after adoption. (Laanti, Salo, & Abrahamsson, 2011). In one study, it was difficult to assess whether the hybrid method that Intel eventually developed was superior to either XP/Scrum or traditional methods (Fitzgerald et al., 2006). Barlow maintains that the success of an agile team could be determined as a function of the density of the project team's advice network, moderated by the cost of maintaining informal relationships (Barlow et al., 2011). However, this success factor is only at the team level as opposed to the organization level and does not provide a link to agility in the marketplace.

From a research perspective, links between social interaction and project outcomes such as budget, schedule, and quality are subjects of ongoing research. (Cao, Mohan, Peng, & Ramesh, 2009).

Performance and Limitations of Agile Methods

The following section outlines the positive contributions of agile methods to information systems development, as well as the shortcomings found during their implementation and management.

There is almost no question as to the positive contributions of agile development methods. While at times controversial, they are being adopted in one form or another worldwide across a variety of contexts. Agile methods are here to stay, and the perception of the impact of agile methods is predominantly positive (Laanti et al., 2011). Much of this successful adoption

has been due to the impact agile has had on project performance. For example, agile has been proven to reduce defect density by a factor of seven and allow projects of six to twelve months in duration to be delivered ahead of schedule with high quality (Fitzgerald et al., 2006; Schatz & Abdelshafi, 2005).

The literature has shown that many negative perceptions of agile do not hold up under scrutiny. Contrary to popular belief, agile is not an undisciplined approach. In fact it has been found to require just as much discipline as traditional methods (Fitzgerald et al., 2006) (Schatz & Abdelshafi, 2005).

Agile's contributions are not limited to project management and software development improvements. It has also been shown to create improved job satisfaction, productivity, and increased customer satisfaction (Dybå & Dingsøyr, 2008). The practices of sprint planning, daily standups, and retrospectives have been shown to help people function better as teams (McHugh, Conboy, & Lang, 2012).

Additionally, agile methods have been found to positively influence quality, especially in highly turbulent markets where requirements often change. This has been especially true in situations where high outcome controls are used, such as established standards for evaluating project performance. It has also been found to lower software complexity in rapid changing environments (Maruping, Venkatesh, & Agarwal, 2009).

Despite all of its contributions, agile methodology is not a panacea. Agile can be difficult to introduce to large, complex projects, and having continuous customer input can be unsustainable for long periods (Dybå & Dingsøyr, 2008). Agile can also generate obstacles with decision making, such as commitment, conflicting priorities, unstable resource availability, ownership, implementation and empowerment. These can result in the absence of strategic

roadmaps, lack of team engagement, and an ever growing backlog of delayed work from previous iterations. This backlog is often referred to as technical debt (Drury, Conboy, & Power, 2012). The high level of empowerment that an agile team often has can result in groupthink or Abilene paradox (Abrahamsson et al., 2009). Determining solutions to these obstacles is critical to agile project and team success (Drury et al., 2012).

For the individual developer, agile presents another set of challenges. Gold plating, or adding features that the customer never asked for, can be a common issue because programmers like to be creative, and the autonomy provided by agile gives them this latitude (Baskerville & Pries-Heje, 2004). Agile techniques such as Scrum can put developers "on the spot" and cause them to fear exposure of their weaknesses. Placing two developers to code together, referred to as pair programming, can be one technique to guard against such shortcomings.

Also, agile developers must wear many hats, often playing the roles of coder, tester, architect, customer, QA expert, and so forth. It can be difficult to find people with such a broad skill set, particularly when it comes to the interpersonal or business skills necessary to function in these roles. Taking on such responsibilities often makes it difficult for developers to hone the specific skills required for their job, which may inhibit promotion. Mitigating these issues requires agile specific policies across the organization. Agile values and principles need to be integrated throughout, and periodic assessments of a team's agility should be conducted using an assessment framework based on agile goals, as opposed to practices (Conboy, Coyle, Xiaofeng, & Pikkarainen, 2011).

Agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, instead of formal documentation. As a result, unapparent shortfalls in this knowledge

can lead to significant mistakes. This may be exacerbated in the teams are large. Agile has been found to become unwieldy with teams beyond fifteen to twenty developers (B. Boehm, 2002).

Distributed development teams can have their own unique set of conflicts related to agile, including lack of team cohesion, people versus process controls, communication, and formal versus informal agreements. However, studies have shown that much of this can be mitigated through knowledge sharing, intensive communication, trust, and a practice of continual improvement (Ramesh, Cao, Mohan, & Peng, 2006). Such issues can further be resolved with the establishment of contextual ambidexterity through the balancing of formal structures with flexibility, trust with verification, and process assimilation with quick delivery (Ramesh, Mohan, & Lan, 2012).

Iterative agile development can often cause too much focus on short-term deliverables, which can create situations in which the resulting end product is unshippable. Dedicated sprints must often be created to fix bugs, due to the time-boxed nature of sprint planning. Lack of focus on non-functional requirements, such as scalability or long term maintainability, has always been a challenge for the agile organization. Burndown charts do not sufficiently communicate remaining work for a release because of the changing nature of requirements. Accurate reporting requires a greater level of discipline within the teams to provide regular and accurate feedback (Schatz & Abdelshafi, 2005). It has also been noted that there is a tendency to underestimate tasks in agile because even for experienced developers, estimating the unknown can be difficult (McHugh et al., 2012).

Finally, agile increases the risk of overemphasizing functional requirements, incomplete or inadequate requirements, and inadequate design (Ramesh, Lan, & Baskerville, 2010). This makes the management of agile particularly challenging at the organizational level where

requirements are emergent rather than specified up front (B. Boehm, 2002). Misunderstanding user requirements has been cited as a major cause for software quality problems in agile as well as other methods of development. Studies have shown that such quality problems can better be solved by improving communication, rather than testing (Sue et al., 2012). Ensuring that such communication occurs can be a challenge in an environment with no Stage Gate process to provide checks and balances.

Limitations of Agile Research

As previously mentioned, agile software development methods have only come into practice during the last decade. As a relatively new concept in software development, much of the current literature lacks clarity, theoretical glue, and parsimony. In addition, much of this literature has limited applicability to various contexts (Conboy, 2009). Most importantly, the current body of research lacks clarity with regards to what agility is, its adaptability, and how it is deployed in practice (Abrahamsson et al., 2009).

Researchers have noted that there is an overall need to improve the rigor of agile research (Abrahamsson et al., 2009). In many such studies, research methods were not well defined and weaknessess regarding bias, validity, and reliability were not addressed. Employment of applicable theoretcial frameworks are rare. Data collection and analysis processes are often poorly defined and the "current contribution of agile research remains low and uncertain" (Abrahamsson et al., 2009).

Agile methods, especially in the early years of the *Agile Manifesto,* have been used primarily in small teams at younger companies. As a result, there have been few studies in mature teams or teams in larger organizations (Dybå & Dingsøyr, 2008). Research studies framed in known general contexts, such as embedded development, are also lacking

(Abrahamsson et al., 2009). This is compounded by a dearth of research in post adoption contexts and innovation (Abrahamsson et al., 2009).

More case studies are needed to evaluate control patterns in different contexts. For example, it has been found that multiple categories of control, including both formal and informal, are needed in large distributed development contexts (Persson, Mathiassen, & Aaen, 2012).

Although there is a significant and growing body of research on agile, many aspects are yet to be explored, particularly outside systems development at the organizational level (Conboy, 2009). The extent to which various stakeholders inside and outside the organization contribute to agility and agile teams is yet to be investigated (Conboy & Morgan, 2011). This is largely due to the fact that agile has traditionally been championed from the bottom up, being implemented at the development team level with few if any agile concepts being adopted upstream (Abrahamsson et al., 2009). Barlow cites that team culture, top management support and alignment with organizational strategy were typically not included in determining an approach to agile adoption (Barlow et al., 2011). However, Highsmith maintains that true agility requires assimilation of agile concepts into all aspects of the business including determining success factors (Highsmith, 2010).

Abrahamsson tells us "there is a poignant need to identify rigorous ways with which agility can be assessed" (Abrahamsson et al., 2009). Examples cited included determining the decline of agility over time, across projects, and at the organizational level to identify improvements. Most importantly, there needs to be a way to bridge the understanding of agility to system development success (Abrahamsson et al., 2009).

To bolster these claims further, Conboy asserts that "mechanisms for scanning the project landscape should be incorporated into project management practices in agile organizations." The same study further asserts that "project managers need to be aware that an information systems project is no longer a local matter that can be treated as a closed innovation isolated from the rest of the organization." Such projects should be seen in light of other projects within an organization" (Conboy & Morgan, 2011). Similar studies have drawn a clear distinction between "doing" agile and "being" agile. True agility requires significant cultural and procedural changes within an organization as well as a new thought process. A prerequisite for information systems development agility has been stated as the practice of *mindfulness routines* as *organizational routines*. Mindfulness routines have been described as the practice of "gathering new information from multiple perspectives via self-assessment and reflection to promote continuous creation and refinements of organizational routine performance" (McAvoy, Nagle, & Sammon, 2013).

A practice perceived as new by its adopters, such as agility, can be considered an innovation (Rogers, 2003). Agile methods are often viewed as process innovations of an information systems development organization (Wang, Conboy, & Pikkarainen, 2012). The assimilation of agile has been conceptualized using innovation diffusion as a lens, concentrating on the stages of acceptance, routinisation, and infusion (Wang, Conboy, & Pikkarainen, 2012). In a software context, agility is affected by the extent of innovation in base technologies as well as process innovations in complementary assets (Lyytinen & Rose, 2005).

Software organizations organize themselves differently during different innovation periods while they decide to explore fast or deliver fast. They control their focus on agility on

how good they want to become in managing technologies in different innovation phases (Lyytinen & Rose, 2005).

Current research needs more careful constructs for agility and other process features. Variances have been found in process features, across phases, and between companies due to varying focus on exploration or exploitation. There is a poignant need to explore other factors than just an organization's learning focus to establish causal explanations of agility in organizational contexts (Lyytinen & Rose, 2005). In summary, systematic and insightful understanding of agile methods in use is yet to be achieved (Wang, Conboy, & Pikkarainen, 2012).

Chapter 5: Research Methodology

Research Design

The purpose of this study was to answer the question of *how* agile processes are orchestrated in an embedded systems environment. The study did not require any behavioral controls, and it focused on contemporary events. The assimilation of agile methodologies within embedded systems development is indeed a most contemporary phenomenon that requires in-depth analysis in a real-life context. The boundaries between these are not clearly evident, and there are many more variables of interest than data points due to the complexity of agility measurement. These are all key characteristics of research conducive to a case study approach (Yin, 2009). In addition, this case study approach was conducted in the interpretive tradition of information technology studies (Klein & Myers, 1999). As a result, focus was placed on the participants descriptions of software development practices and their work related to them.

A grounded theory method of data analysis was employed. Grounded Theory is a qualitative research methodology that does not begin with a theory; instead it starts with an area of interest and allows the theory to emerge from the data. Strauss and Corbin define it as "a qualitative research method that uses a systematic set of procedures to develop an inductively derived grounded theory about a phenomenon (Pozzebon, 2011). Research results consist of grounded theories discovered inductively by collection and analysis of qualitative, empirical data (Baskerville et al., 2011). Grounded theory is most appropriate where research questions are descriptive and explanatory, and the field of phenomena is not well studied and lacks a substantive body of theory (Galliers, 1991).

Two variations on grounded theory can be found in the literature: Glaserian and Straussian (Pozzebon, 2011). While the Glaserian method advocates an unstructured approach

for data analysis and theory construction, Straussian provides a well-defined set of procedures for applying the method (Pozzebon, 2011). Although both approaches to grounded theory can be found in information systems literature, the Straussian method is much more prevalent. Its structured design also better lends itself to a doctoral dissertation (Pozzebon, 2011). It is for these reasons that this study employs Straussian grounded theory as its research methodology. The methodology herein is based on Strauss and Corbin's seminal work *Basics of Qualitative Research: Grounded theory procedures and techniques* published in 1990 (Strauss, 1990).

This research effort consisted of a singular case study. One of the often-cited limitations of using one case study is its lack of generalizability, as the data collected is often specific to the particular situation at a particular point in time (Fitzgerald et al., 2006). However, quality with interpretive case studies is defined by the plausibility of the story and the argument it presents, as opposed to validity and reliability found in positivist studies (Klein & Myers, 1999). More importantly, the rich detail provided by case studies is considered more valuable than generalizability (Yin, 2003).

Data Collection

The case chosen for this study was an embedded systems development organization in the power utility industry. Beginning five years ago, this organization instituted a mandate to integrate agile development methodologies enterprise-wide. Although the implementation and evolution of agile adoption is known to many of the participants and adds context to the study, this research is focused primarily on the post-adoption state that the business currently finds itself in.

Participant observation, interviews, and documentation were all used as data sources for this effort. Participant observation was used because the first author has been an employee of the

organization for almost five years. As a result, he has been a regular participant in several projects including the continuous development of the organization's primary product line. He is and has been a regular attendee for virtually all meetings and activities relating to these projects, and is intimately familiar with, not only with the development processes and procedures themselves, but also how they have evolved over time. During the past five years he has participated in the organization's adoption, assimilation, and ultimate hybridization of agile methodologies.

The second source of data consisted of semi-structured interviews with managers and lead architects involved in managing agility. As an employee of the organization being studied, the first author was able to identify and access the interview candidates using his knowledge of the organization. Roles were selected based on their knowledge of and impact of the orchestration of agile processes and product development. This selection method was based on the criteria for *key informants* as outlined in Klein and Meyers (Klein & Myers, 1999). Four key roles were identified as those regularly involved in these activities. These included product managers, project managers, engineering managers, and technical architects. These four roles were found to have the most hands-on impact in managing agility for the products, processes, people, and technology in the organization. Embedded systems development in this company consisted of three primary domains: hardware, firmware, and software. Candidates for each of the four roles in all three domains were selected, resulting in a total of twelve interviews.

As previously mentioned, the interviews were semi-structured in format. Interview questions were informed by the literature using the Strausserian approach to grounded theory development (Strauss, 1990). The candidates who were interviewed were assured of anonymity. Interview sessions lasted on average about an hour in length. The interview data gathered was

recorded, transcribed, and coded using Nvivo software. The first author's familiarity with the organization and the interview candidates had a positive influence on the results. For example, the first author had unfettered access to select any candidates that were willing to participate. In addition, interviewees seemed comfortable and candid with an interviewer they knew personally.

Documentation served as the final source of data which included archival data of agile processes and procedures, meeting minutes, and project artifacts such as feasibility studies. These sources allowed the author to better understand the issues and outcomes of management decisions during project life cycles and to fill in any gaps of understanding with respect to processes, procedures and history.

Data Analysis

Data analysis was performed based on the three commonly used coding techniques in grounded theory research: open, axial, and selective coding (Strauss, 1990; Baskerville et al., 2011). Open coding is a process of analysis that develops concepts in terms of their properties and dimensions. Key tasks involved are the asking of questions regarding data, and then making comparisons. Similarities are grouped to form categories (Strauss, 1990). Essentially, the text was broken down into segments which were compared for similarities and differences. They were then labeled and grouped to form codes. A single code could have multiple text segments. During coding, 542 codes were created. As an interpretive case study, the data primarily reflected the interpretations that the interview candidates formed about agile process orchestration within their organization and their work relating to it. As a result, the coding categories included both positive and negative views of agility in the organization and described actions taken by many of the interview respondents.

Once the data was broken down via the open coding, it was reconstructed in new and different ways using the axial coding method. Axial coding is a process of relating subcategories to a category through a process of inductive and deductive thinking. Although it too involves the tasks of asking questions and making comparisons, categories that arise from this analytic method are developed in terms of causal conditions, context, consequences, and action/interactional strategies (Strauss, 1990). Through axial coding, six larger categories were developed. The first three categories reflected the company's market agility. The fourth category identified process agility and the organizational context: hybridized agile methodology and embedded systems development. The final two categories described agile orchestration. Sample codes for these categories can be found in Table 1.

The final step was selective coding. Selective coding is the process of selecting the core category, relating that category to others, validating the relationships and completing those which need further development (Strauss, 1990). Strauss and Corbin advocate continuing this grounded theory coding of data until one single category stands out. (Baskerville & Pries-Heje, 2004). Selective coding was considered complete once saturation had taken place. Such saturation occurs once there is no additional data to inform a category and the relationships between the categories have adequate data to support them. The final result is a story line that correctly conceptualizes this core category or primary phenomenon. This story line is the heart of grounded theory. The next section accomplishes this task by presenting the empirical analysis, followed by the presentation of the final theory of the *agile business vortex*.

Chapter 6: Results

In this section, we first describe the institutional context surrounding the orchestration of agile processes within an embedded systems development organization. Then we describe the three main constructs: generation of *market agility*, the development of *process agility*, and the *orchestration of agility* within the embedded systems development. These constructs are then rolled up into the grounded theory of the study which is conceptualized by a comprehensive illustration combining all of the constructs into one end-to-end view. Table 1 provides an overview of these constructs, their major categories, and code examples for each. Each section includes a table of elements for each category. For a more detailed view of how all categories and elements are linked, refer to the category diagram in Appendix A.

Institutional Context

The institution in this study is a company in the business of developing embedded systems devices for use at power utilities. It is an international corporation employing over 5,000 individuals worldwide and has recently become a subsidiary of a major electronics firm. It is important to note that the company's contracts with its customers are largely based on the number of embedded devices that they sell. These devices have a long lifespan of between ten and twenty years. As a result, once a customer is *taken*, it puts them out of the market for quite some time. The company has developed a strategic direction of establishing itself as a market leader by "grabbing up" market share before its competitors, and it has been largely successful in doing so. Although the firmware and software aspects of the embedded systems are critical to the overall functionality of its products, company profits are primarily driven by how much hardware can be manufactured and sold to a given client. Because of this, it was critical to include hardware as well as software and firmware domains in this study.

It should also be noted that the subject of this study was the result of several mergers over the years, with a large number of disparate technologies, processes, and procedures that have slowly evolved and converged over time. As it was largely a hardware development firm in the beginning, the company's projects are grounded in a waterfall process, termed *new product introduction*, that it has inherited and largely maintained over the last seven years as agile methods have been integrated. As with many similar organizations, the company has gradually adopted and adapted agile methods into the enterprise, with the highest degree of adoption in areas with the highest rates of change, such as software and firmware.

Three years ago the company embarked on an initiative to adopt agile and Scrum methods in earnest, which was the third iteration of the company's agile adoption process. This phase is now considered complete, with the result being a hybrid agile implementation across the enterprise with varying degrees of absorption per domain. At this time, the company has no further plans to make any process improvements or changes with respect to agility or agile methodologies.

Category		Sample of the Codes within a category
Market Agility		
Market Pressure		Competition, Government Regulation
Product Genesis		Requirements Comprehension, Dynamic Priorities
Process Agility		
Hybrid Agility		Software, Firmware, and Hardware
Agile Orchestration		
Interconnections & Interactions		Dependencies, Interdependencies, Linkages, Decision Points, Status Points, Touch Points
Making Adjustments		Customer Negotiation, Resource Adjustment, Scope Adjustment, Constant Re-assessment

Table 1: Key Constructs and Major Categories from Axial Coding

Market Agility

Market agility, in the context of this study, is the ability to adapt to market pressures via product genesis. In essence, the business responds to market pressures with a product roadmap, based on its understanding of customer needs and the internal capacity to meet those needs. The resulting product scope, or roadmap, creates *business momentum*, which *process agility* attempts to match with the *systems release*. This section begins by describing the elements of market agility, starting with its key driver, which is market pressure. This is followed by a description of product genesis, which is the company's market agility response to such pressure. The section ends by summarizing market agility with the culmination of business momentum.

Market pressure in this study was found to be composed of six elements. These included four *pressure drivers*, which are market share, customer base, government regulation, and competition, followed by two limiting factors or "governor elements" that served to keep the pressure in check. While most of these elements within market pressure are outside the control of

the organization, there are some aspects that it can influence. These elements are outlined in Table 2 below, followed by descriptions of each respectively.

Elements of Market Pressure	
1. Market Pressure	1.1 Market Share
	1.2 Customer Base
	1.3 Government Regulation
	1.4 Competition
	1.5 Governing Market Pressures: Strategic Direction
	1.6 Customer Appetite

Table 2: Elements of Market Pressure

Although gaining *market share* is important to any corporation, participants in this study brought forth characteristics that made this context particularly challenging. The embedded devices in this case study consist of new smart grid metering technology. These devices have a long lifespan, and are often sold in large numbers. Customer contracts tend to be long term, often spanning decades. Installation of the devices takes place over a period of several months, and once they are installed customers expect maintenance over the life of the product. The customers in question are power utilities which are often the only service providers in their respective areas. As a result, there is a limited amount of *territory* to be had, and once that territory is sold, it is out of the market for a very long time. These factors have created what one respondent termed a *land grab* situation. Vendors are under exceptional pressure to grab up as much market share as possible before it is all gone. Those who do not are not expected to survive in the business for long, or will at least face long-term marginalization in the market.

This phenomenon is analogous to the *gold rush* situation noted by Vigden, which can create very high market turbulence (Vidgen, 2009).

Respondents also noted the relatively small, intertwined *customer base* as being a factor. Although power utility companies are often large government regulated entities, there are relatively few of them. Utilities are part of a small tight knit community that readily exchange information about their vendor experiences. Recommendations from utility customers can not only help sales, they can make or break a vendor in the business. This gives customers a great deal of leverage when it comes to getting what they want out of the product.

In addition to this limited customer base, strong *competition* between vendors was also felt by many of the respondents. Other vendors were believed to be more agile and nimble in some cases because they did not have the baggage created by numerous mergers and acquisitions over the years. Competitors were believed to have the ability to respond to the market just as quickly with equivalent feature sets and embedded device support. The respondents felt that this resulted in a constant battle of who could provide the richest feature sets in the least amount of time. As one respondent explained:

> *"They [our competition] go to the customer and say, 'hey, these guys don't have this latest and greatest [feature but] we have it,' so agility is certainly important." "Since everybody is responding to the market, if you're the one who is doing it quicker, it helps your business."* Hardware Project Manager

Governments were also found to be a market driver. Being an international company, this organization was subject to, and worked with, a number of different governments all over

the world, and even domestically, it was often faced with issues within different states, counties, and municipalities. Although government funding often helped drive the adoption rate of the smart grid technology that the company sold, sudden changes in regulations could have a significant impact on required feature sets and quality standards:

> *"In a regulated environment, we find that we are reacting to the whims of government change and having to adopt changes to the products to head off being excluded from bidding on future projects."* Software Project Manager

The utility industry is highly regulated. Mistakes in billing and meter reading have garnered a great deal of media attention, and hence, governments have passed strict regulations on billing requirements and feature sets to ensure accuracy.

Despite the pressures of gaining market share in a turbulent environment, intertwined customer base, government regulation and stiff competition, participants agreed that market pressure did not go on unchecked. The organization was adept at employing methods of maintaining a good mix of responsiveness and control. One of these methods was the establishment of a *strategic direction*. In order to get products to market as fast as possible, close customer collaboration was often required. The company made strategic decisions to concentrate on those customers who were willing to support this high level of collaboration by acting as testers for new technology in exchange for being the first in the industry to obtain the latest feature sets. Such collaboration allowed participating customers to influence the technical direction of new features. This approach in turn allowed the organization to get products to market faster along with the added benefit of conserving resources by using the customer as an

extended quality assurance team. Although such customers had to deal with extensive defect rates, they were compensated by receiving higher levels of service. By contrast, customers who demanded proven products out the gate were often ignored. In essence, the company made strategic decisions to adjust customer focus to those willing to collaborate heavily in order to maintain its competitive edge. One respondent summarized this concept as follows:

> *"Some customers are just simply adamant that they get a proven product, they don't want to deal with our problems and, you know, 'It had better come to them tested or, you know, there will be repercussions.' Those are customers that we tend, frankly, not to focus on. If they are going to take that approach that's unrealistic, then they'll get older product and they'll get less attention, because, again, that land grab is the strategic priority."* Software Product Manager

Another control on market pressure is customer appetite. Although customers willing to collaborate are going to get the latest features and the most attention, there's only so much innovation that they can handle at a given time. Each release must be qualified by the customer before they can accept it, and the continuous release nature of agile development methods can create more work for them. As one respondent put it:

> *"Customers don't have an appetite for numerous system releases just because, again, the complexity of the system release and the level of integration effort and expense on their side to qualify."* Firmware Product Manager

In summary, the findings show that market pressure is the key driver of market agility. This pressure is created by the drive to achieve market share, the size of the customer base, competition, and in the case of this study, government regulation. At the same time, this pressure is governed by customer appetite and the company's own strategic direction. These elements outlined in Table 2 illustrate how this response to and control of market pressure creates an interesting balance that allows the business to respond to customer needs and *stay just ahead* of the competition without exceeding its capabilities.

The organization responds to market pressures through the creation and evolution of its product line, or *product genesis*. Product genesis is essentially the organization's market agility response to market pressure. At a high level, it consists of the prioritization and comprehension of requirements, followed by scope negotiation and development of the product roadmap. Table 3 below illustrates the elements that comprise it.

Elements of Product Genesis	
2 Product Genesis	2.1 Establishment of Market Timing
	2.2 Dynamic Priorities
	2.3 Requirements Comprehension
	2.4 Decomposition
	2.5 Dropping Requirements
	2.6 Scope Negotiation
3 Business Momentum	

Table 3: Elements of Product Genesis

The process of product genesis begins with the introduction of new requirements. Product managers noted that they serve as the first entry point for these new requirements and

are the primary interface to the customer. They play a large part in managing the components of market agility by gathering the initial requirements, establishing priorities, contributing to requirements comprehension, and negotiating scope with both the customers and internal stakeholders. Their first step, according to the respondents, is to create business cases from the initial requirements gathering, and work to determine where in the product roadmap it should fit. This is referred to as establishing the *market timing*. Market timing ensures that the product roadmap is in alignment with current market pressures, as well as internal needs. Respondents stated that the establishment of such timing was subject to a number of internal as well as external factors, which included dependencies on other development efforts, organizational priorities, resource availability, and budget.

Once market timing is established for a requirement or feature set, it is then *prioritized*. This prioritization changes *dynamically* as requirements and their impacts become better understood. This understanding is an iterative process of requirements comprehension and decomposition with the input of both product management and engineering. Requirements are decomposed into components which are typically translated into user stories. As these stories are discussed, reviewed, and estimated, the understanding or comprehension of these requirements changes. The priority of the requirements can therefore change as a result of this iterative analysis. This phenomenon was referred to by respondents as "bubbling up" the requirements. As this dynamic priority becomes better established, requirements are then accepted, dropped, or postponed, and the scope for the systems release (and therefore the overall product roadmap) becomes more solidified. The scope of the latest systems release in essence *bubbles up* to the top of the product roadmap. Such prioritization is dynamic throughout product genesis.

So how exactly does prioritization occur? Respondents stated that prioritization is a constant "push and tug" process between the business and engineering, or as in the context of this study, between market and process agility. It is both dynamic and iterative and becomes better defined as *requirements comprehension* and decomposition progress. Respondents stated that product managers compete with each other as well as engineering stakeholders to bring visibility to their priorities. The dynamism of these priorities is somewhat dependent on their foreseeability and perceived impact. In summary, prioritization is a process of assessment and reassessment of features against company strategic direction, the value of deals coming in, and internal competition between product managers. This process evolves as requirements and their impacts become better understood.

If requirements comprehension is key to the evolution of priorities, then how does it occur? Respondents stated that requirements become better understood as they are broken down through *decomposition*. Decomposition was defined by respondents as the process of increasing understanding by reducing complexity. Complexity is reduced by breaking requirements down into manageable chunks so that the technical, financial, and product implications are clear to the stakeholders. The output of this process is typically a series of user stories that can be fed into development sprints once they have been reviewed and accepted for a system release. Once a requirement is decomposed, additional dependencies, requirements, resource, or budget needs often become more apparent. This information is then fed back into the prioritization. Ultimately, the business seeks to understand how much capacity a specific requirement will need, its technical impact, and value add to the company's product roadmap and customer base. This process is a vital input to product genesis, because it dictates what features may or may not make it into a specific system release. It is the way in which product genesis *right-sizes* itself as

requirements and business needs become better understood and agreement with the customer (or market) is negotiated. The diagram in Figure 1 illustrates this cyclical process.

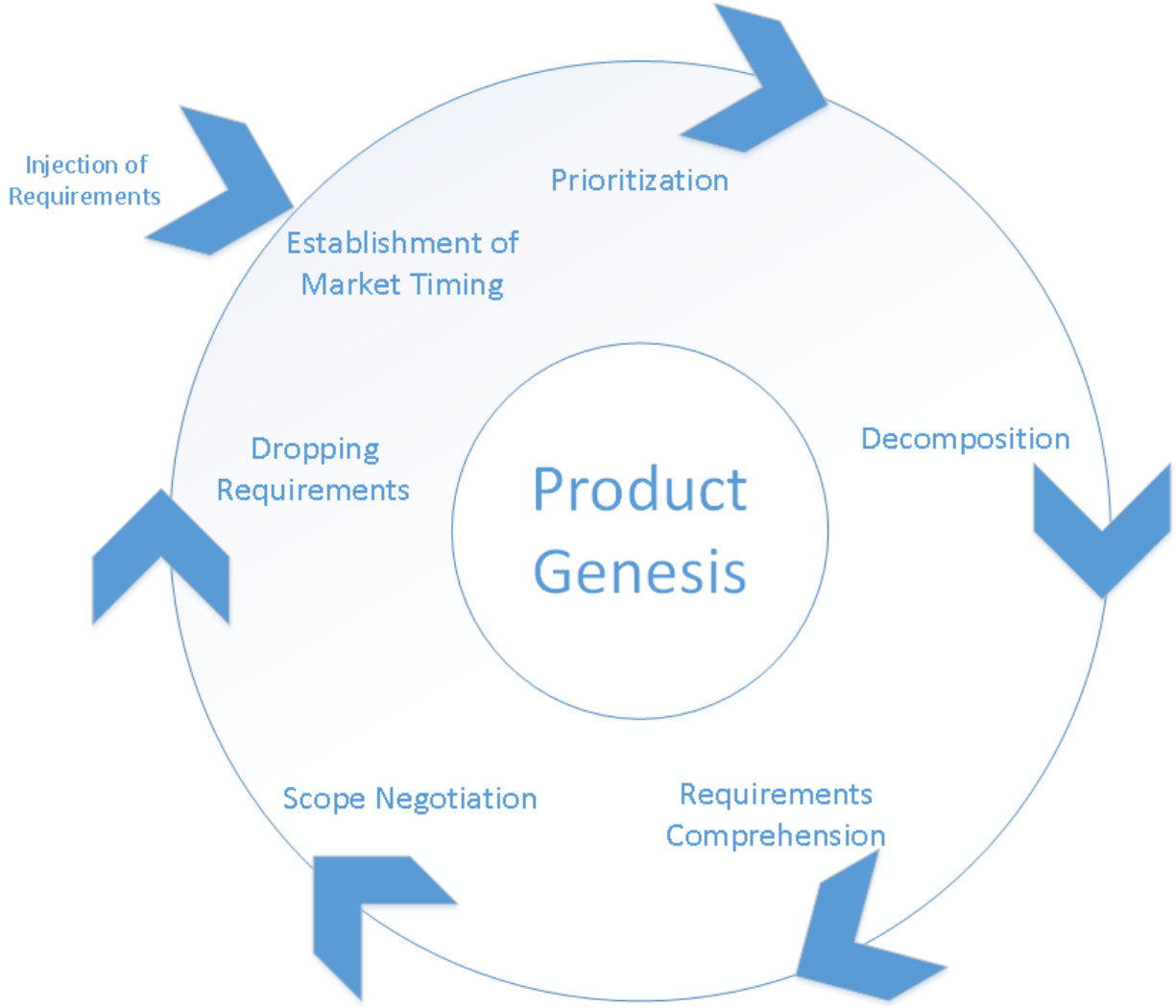


Figure 1: The Evolution of Product Genesis

In the end, company culture plays a large part determining the level of market agility. According to one software product manager:

> *"So, you know, it is probably a cultural strength, and the reason for our success is that we are used to change and it's not a cliché."*

> *"But the aggressive nature of our expansion and our corporate parent pushes us in that direction and fortunately the culture can stand it because it's used to it in North American business. There are jobs I've been in where the cultures couldn't stand it and that usually ended up with a bunch of people quitting when something moved to a different mode of operation or went from domestic to international business."*
>
> Software Product Manager

To summarize, we have shown that market pressure is driven higher through the need to gain more market share. Such pressure can also be driven by a small tightly knit customer base or community that readily exchanges detailed information about the vendors, their offerings, and customer experiences. Government regulation can drive market pressure in this industry through government mandates to utilities to adopt certain technologies, enforcement of billing standards for embedded device performance, and government financial incentives. These market pressure drives are tempered by a company's strategic decision making to accept or ignore certain pressures, and the appetite for customers to accept innovation at a given time. Product genesis is the business's *agile response* to this market pressure. It begins with establishing market timing for a set of feature functionality, and dynamically prioritizing these feature sets as they go through a requirements comprehension and decomposition process. As this process arrives to completion, some features are dropped or retained as the final scope is negotiated. What remains is the scope and timeline that the organization strives to "reach." This scope and timeline creates *business momentum* that the organization often finds itself chasing after.

While the previous sections illustrated the components of market agility, the next section includes the output, namely the product roadmap and the momentum it creates. This roadmap is comprised of a series of system releases each of which have a scope and timeline, as created via product genesis. System releases are essentially complex embedded systems that are comprised of software, firmware, and hardware components released in tandem.

In the world of physics, momentum is mass multiplied by velocity. Business momentum in the context of our study is the scope of the release or product roadmap (release mass) multiplied by the velocity or timeline in which the organization is attempting to achieve it. The direction in which this flows is in the direction of technological innovation. As market pressures increase, this business momentum can be sensed within the organization and can feel as though it is increasing or building over time. It begins with an aggressive initiative to gain market share, which feeds into the product genesis and results in the size and speed of the system release.

Respondents noted that as this momentum builds, it creates a ripple effect that can be felt throughout the organization. As it gains speed, managers, developers, and engineers may feel as though they are always "behind the curve," never having enough time to build in robustness or form long lasting and architecturally sound solutions. The following comments illustrate this sense of momentum:

> *"It seems to me that we're always behind the curve, and we've just got to get it done, and there's not enough time to do architectural work and look for – you know; make sure it's being done correctly and for long-term extensibility."* Firmware Manager

> *"We don't usually get the time, just don't have the resources, at least not for the last couple of years, to really – to look ahead and evaluate new technologies. It feels like we're always a little bit behind the curve and reactive."* Hardware Architect

Despite all of this, the development organization attempts to rise to the challenge and match this momentum. As one respondent explained:

> *"There will be a call for extra hours, weekends to try and make it. I wouldn't say we change the deadlines, we just roll over them and we get in the 'as soon as it's done' mode. We try to condense when things change."* Firmware Architect

The "condensation" expressed above is one example of how the organization attempts to match this challenge. Although market agility sets the tone through product genesis and the momentum it creates, the development organization uses its own form of agility, described in the next section as *process agility*, not only to match the momentum but to influence it as well.

Process Agility

In this embedded systems organization each of the three domains, software, firmware, and hardware are capable of releasing independently and at different speeds. At some point, however, all three domains must work together to create a *system release*. A system release is one where components of all three domains are developed, tested and released together as one cohesive product. Doing so often stretches the capability of the organization to its limits. It is this crucible where process agility is flexed or adjusted in order to reach the same point of

momentum that the business has been leading, via market agility. The organization makes this happen via hybrid agility.

Hybrid agility can be defined as a delicate balance of agile development methods and Stage Gate processes. For example, in a hybrid vehicle, electric power is utilized as much as possible to maintain economy, but it is augmented by gas motorization when extra power is needed. Hybrid agility makes use of its stage gate and agile scrum components in much the same way. Agile scrum methods were employed across domains to allow the development organization to "rev high" when needed, while at the same time stage gate components served as a sort of "throttle" for this capability. Table 4 outlines the elements used to create this balance. It illustrates how agile development methods allow the organization to *stretch* or "*reach* when it needs to, while the Stage Gate aspects keep the entire process in check. Doing so allows all three domains to work together at an optimum level to achieve system release.

Elements of Hybrid Agility		
4 Hybrid Agility	4A Software	4A.1 Employs mostly Agile/Scrum Methods
		4A.2 Serve as the Early Responders
	4B Firmware	4B.1 Employs some Agile Methodologies
		4B.2 A Shared Resource: The Middle Domain
	4C Hardware	4C.1 Employs waterfall process
		4C.2 Prototyping
		4C.3 "C-Level" Projects
	4D Customer Management	4D.1 Managing Expectations
		4D.2 Customer Negotiation
5 System Release		

Table 4: Elements of Hybrid Agility

Respondents indicated that each domain approached agility differently. The claims in Table 4 above outline these differences from one domain to another. One manager attempts to explain why some of these differences exist:

> *"The problem we found in agile was morphing and meshing that set of work to the waterfall methodology for hardware development"* Software Product Manager

Figure 2 below attempts to illustrate the characteristics of each domain and how they fit together. It is important to note that these "agile characteristics" which vary from one domain to the other are largely unique to the embedded systems context. For example, the software domain acts as

the “early responders” for high priority issues when the other domains cannot respond as quickly. Firmware is a shared resource for both software and hardware domains, and hardware will often employ rapid prototyping or fast track projects to keep up with its more nimble domain cousins. These characteristics enable the three domains to work together as one cohesive unit. Further, all of the agile characteristics within each domain are collectively grounded by the stage gate components of the process. In addition, customer management activities are typically performed with the input of all three domains acting as one cohesive unit when communicating with the customer.

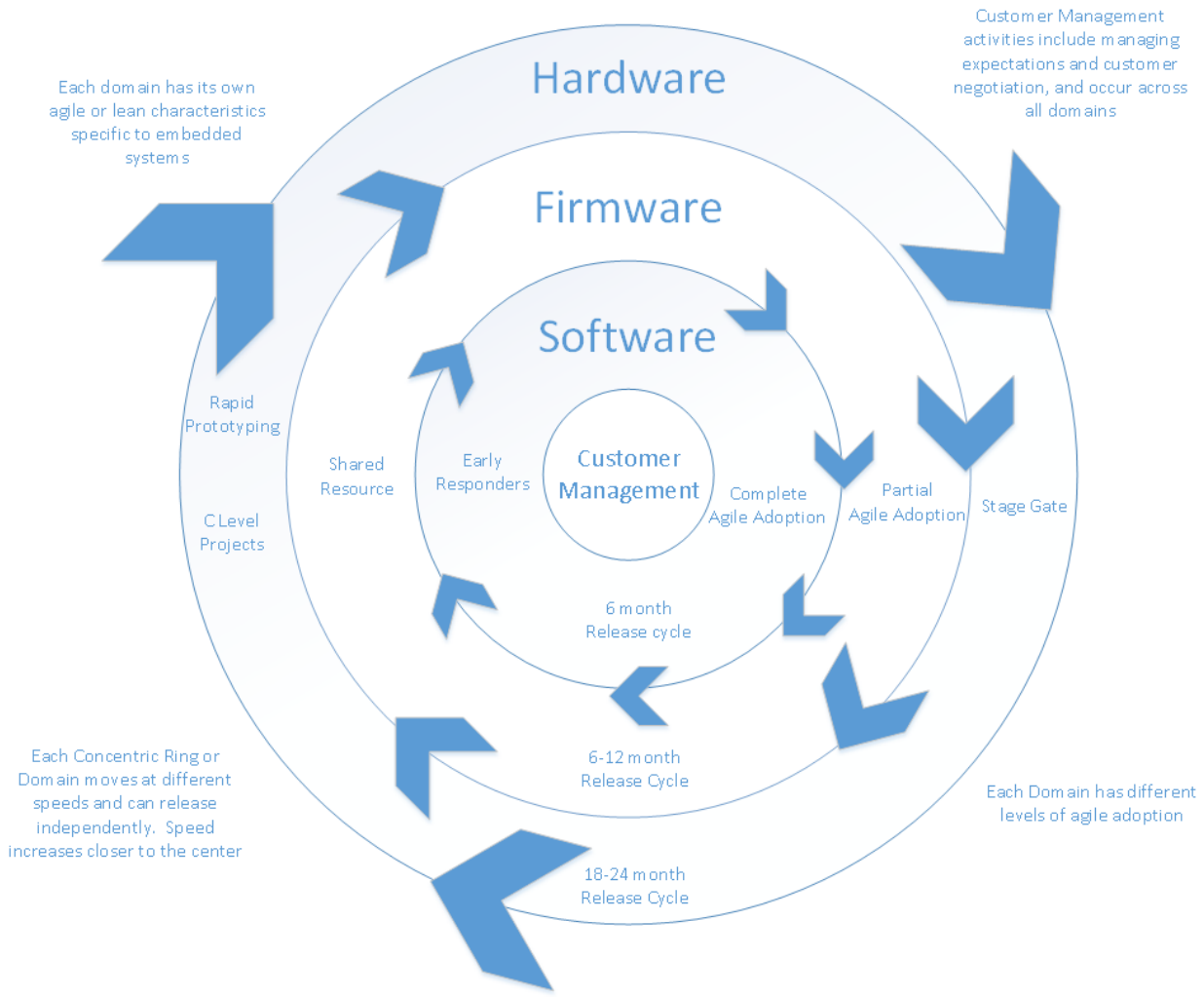


Figure 2: Hybrid Agility in Embedded Systems: Key Characteristics

Although merging agile and Scrum with the Stage Gate methodology may have been driven in part by the need to incorporate hardware projects more effectively, it also served as a series of *sanity checks* for the organization as a whole. The purpose of this sanity checking is to ensure that the system release matches what the business needs. It is how process agility lines itself up with market agility at a specific point. In short, Stage Gate acts as a control or checkpoint on agile methods. One product manager explains these *toll booth* characteristics:

> *"[The merging of agile methods] to gate-driven process is more or less like a toll booth. Before you go onto the next section of road, do you have the right fare to get through? And did you get the right checks of the requirements? Did you get the right financial backing? Did you get the right details in the technical pieces and how you are going to get to the next toll gate? That is our NPI gate-driven methodology."*
> Software Product Manager

This kind of sanity checking is often necessary in an embedded systems environment due to the complexity of the solutions, interdependencies, and the need to eventually roll components up from all three domains (software, firmware, and hardware) into one comprehensive system release. As one firmware manager explained:

> *"That complexity [of] firmware, the head-end, and the hardware in order to release it is what contributes to the waterfall methodology of a system release."* Firmware Product Manager

The next three sections describe the domains of software, firmware, and hardware within an embedded systems environment, and their role in hybrid agility. Yet another element of hybrid agility is customer management. Although product management may serve as the primary interface to the customer initially, the engineering organization is not without a voice. Respondents noted that the organization's *voice*, as well as the business side, was a critical component to hybrid agility success.

As with the other domains of firmware and hardware, software can release independently, but is also linked to the other domains. Respondents repeatedly noted that the software domain had adopted the most agile development practices. These included regularly scheduled sprints,

Scrum meetings, retrospectives, and agile methods for requirements management and estimation. Software development in an embedded systems environment can be just as conducive to iterative agile development as software alone, with the exception that it has some constraints or linkages to the other domains from time to time. This is due to the fact that software can be more easily decomposed into testable chunks of code. Because of these factors, the software domain is capable of cutting new releases in an little as six months, compared to hardware which could be up to eighteen months or two years.

Respondents also noted that as the most agile domain of the three in the organization, software often serves as the SWAT team or *early responders* for the company. Whenever there is an urgent need, or even if it is not urgent and merely a process of decomposition, the organization strives to achieve *software only solutions* where it can bypasses firmware and hardware when possible. This tactic contributes to the process agility of the entire embedded system.

Firmware development employs many of the same agile Scrum processes that the software side does, with a few exceptions. Although firmware teams manage requirements through user stories and have regular Scrum standup meetings, the story estimations and sprints tend to be longer and more flexible. This partial adoption of agile is due in part to the fact that firmware development cannot always be broken down into testable, iterative chunks as software can. Respondents stated that size of the "chunks" impacted team velocity and sprint management, making it much more difficult to monitor and manage firmware development in the same way as software. As one manager noted:

"It seems to reach a point where it can't be broken down because it can't be testable – it's definitely not the level of fineness that [software] is." Firmware Manager

From a process perspective, firmware must straddle the organizational divide between the pure agile methodology of software, and the waterfall process of hardware development. More importantly, both software and hardware domains often require support from firmware resources to complete their tasks, which can produce a sort of organizational tension. An architect explains:

"Given that firmware is kind of a shared resource across all these different products and they're following a sprint cycle -- it creates some tension in terms of [interdependencies]" Hardware Architect

Firmware's ability to *stretch* resources in support of the other domains is critical. In many ways it serves as the "glue" which keeps software and hardware connected.

Hardware moves the slowest out of all the domains, with release cycles of up to two to three years in length. Like the other domains, it can release independently, but it is constrained to a certain extent by linkages to the others, particularly when a systems release is needed. Hardware's linkage to manufacturing, longer product life and the associated costs of *spinning boards* makes it difficult to manage requirements in the same way software and even firmware can. As a result, it operates largely within a waterfall context. One of the main reasons cited for this is hardware's inability to drop features as development and manufacturing progress, as cited by a project manager:

> *"With software, you can be agile as you go along and you can drop certain features as needed. With hardware the reason it hasn't been adopted is you can't really do that."* Hardware Project Manager

In addition, the product lifespan of the hardware warrants more extensive quality assurance requirements than the other domains. As a hardware manager explained, this means it cannot flex or compromise in these areas as firmware and software often do:

> *"On the hardware side, we commit 15 or up to 20 years of product life, so since our products are installed, they are exposed to the elements and [must withstand] severe or extreme weather conditions and humidity conditions, so we have to maintain our quality and put a lot of effort in testing and validating"* Hardware Product Manager

Another reason for waterfall methodology is the cost of spinning boards. If new hardware needs to be created due to changing requirements, that can be expensive. This characteristic does not lend itself well to *continuous iterative* development.

Although the hardware domain does not use agile methodologies as the other two domains do, comments from respondents showed that it does contribute to process agility through the use of agile or lean techniques. These include rapid prototyping and by-passing the State Gate methodology when necessary.

Rapid prototyping is one way in which the Hardware domain attempts to *keep up* with the agility of the other domains without outright adoption of agile methodologies or Scrum. In essence, it is exercising an agile capability in contribution to the organization's hybrid approach.

Using this method, the hardware team begins with a working prototype, and then rapidly and iteratively develops subsequent prototypes as requirements change. This is often performed in tandem with firmware development.

The hardware domain has the ability to bypass the Stage Gate process under certain circumstances. These situations are referred to as "C-Level" projects. It is one way in which the hardware domain can suddenly become more agile on demand, as the following comment explains:

> *"There are smaller hardware projects that can be more agile where it's just having to change out one part on a board that's already designed and verify it's good and those . . . don't need as strict following of the NPI process [waterfall]. We call them 'C-Level projects' and they're managed, you know, real loosely. They only have to basically go through two gates, a planning gate and a project closure gate and then the team is allowed to be free in between. We do have many of those type projects and I think they work well if the team plans it well from the beginning. So those are where we're able to take the more agile approach on the hardware side."* Hardware Project Manager

Unlike the firmware and software domains, the hardware domain is managed without the use of sprints, Scrums, or other commonly accepted agile methods. Through prototyping and "C-Level" projects, however, the hardware domain still has an agile or *lean* contribution.

Although the business side serves as the primary communication interface to the customer and the market at large, the engineering organization is not without a voice. Like market agility, process agility not only attempts to reach the momentum set by the business, but

influences it as well. This is done through managing customer expectations and negotiating from a technical perspective when necessary. Such communication is performed by all three domains within embedded systems. As one manager explained:

> *"It can also slow the project down if the customer isn't managed in a way that lets them know 'we're demoing something you asked us to do and here's the result AND the limitations.'"* Operations Manager

Even though agility demands extensive customer collaboration and adaptation, these must be tempered and controlled for the good of the business. The company cannot respond to any and all demands every time. Through managing expectations, the business *grounds* what may often be lofty or unrealistic expectations by the customer with respect to quality and feature functionality.

Not only must expectations be managed with respect to technology and capability, but the deliverable must also be negotiated with the customer. This illustrates that not only does the business have a voice with the customer when it comes to deciding the scope of the systems release, but the development organization does as well. Although the business leads, while the organization largely *reaches*, there is a symbiotic interaction here where the organization may offer more technical input to the roadmap that the business was, or is not capable of, seeing. As a result, the organization and business, or the process and market agility responds respectively, to influence and adapt to customer demands. The following comment illustrates this:

> *"If certain issues are not fixed or if you realize that you won't be able to fix it in time, then they work with the customer to get some kind of a resolution on when that commitment could be satisfied, so in the ideal world you provide everything to the customer, but in reality sometimes you have to go and tell them, 'hey, yes, this is our commitment, but right now it's not working.' With my experience, the customers understand that as long as there is a reasonable time frame to fix or close that gap, I think they always work with us."* Hardware Product Manager

As explained previously, the process agility response of the organization is the hybrid agile implementation. The product of this implementation is the *system release*. System releases are complicated embedded systems developed in a hybrid agile environment. As mentioned previously, they consist of software, firmware, and hardware components released in tandem. The environment in this study has organically adopted the optimum mix of agile and waterfall processes to make the systems release happen.

System releases are strategic as well as practical. Feature functionality that makes the system release can be driven by the desire to gain new business in a specific area, as well as satisfying existing customers. In this way the company can increase business momentum with each release in the direction of innovation. As one product manager explained:

> *"Sometimes, we just need to put things in system releases in order to do something like a proof of concept to gain more business. A lot of times, proof of concepts for bids have tight deadlines around them which could drive their urgency for requirements."* Firmware Product Manager

The system release seeks to match the business momentum that the business side has established. However it is important to keep in mind that both influence each other. Since all three domains within embedded systems can release independently, business momentum can affect each in different ways. For example, hardware may experience a stronger momentum than software, due to the fact that it has a more difficult time adjusting to dramatic change and the "larger mass" of their releases. This in turn may impact tbe scope of such releases. The customer management category within hybrid agility is utilized by the embedded systems development organization to negotiate scope modifications when these situations occur. In this way all three domains are kept to some level of synchronicity within the embedded systems context thru utilization of its hybrid agile implementation.

Now that process and market agility have been defined, the following section will describe how these two categories are managed to achieve the central theory .

Agile Orchestration

Analysis of the data revealed that the activities of orchestrating agility in this case study fall into two main categories: interconnections or interactions and making adjustments. Table 5 below outlines these categories and their elements. Interconnections consist of people interactions and technical connections that communicate, monitor, and synchronize with each other. The enterprise uses these interconnections to make adjustments, thereby bringing market and process agility closer together.

Table 5: Elements of Agile Orchestration		
6 Agile Orchestration	6.1 Interconnections and Interactions	6.1.1 Dependencies
		6.1.2 Interdependencies
		6.1.3 Linkages
		6.1.4 Status Points
		6.1.5 Decision Points
		6.1.6 Touch Points
	6.2 Making Adjustments	6.2.1 Customer Acceptance
		6.2.2 Scope Adjustment
		6.2.3 Resource Adjustment
		6.2.4 Constant Re-assessment

Table 5: Elements of Agile Orchestration

Interconnections are intersection points between different domains within the embedded systems environment. These interconnections can be interactions between people or dependencies based on technology or resources. The major categories that arose from the data included dependencies, interdependencies, linkages, decision points, status points, and touch points. Dependencies and interdependencies are involuntary connections that are forced due to the nature of the technology and the product(s) being developed. The remaining connection types are voluntarily initiated connections created by the organization to manage the first two. Table 6 provides a summary. The following sections describe these different categories and their relationships to each other.

Understanding Interconnections and Interactions in Hybrid Agility		
Connection or Interaction Type	Formal or Informal	Definition
Dependencies	Informal	One domain has a technical or resource dependency on another.
Interdependencies	Informal	Two or more domains have technical or resource dependencies on each other.
Linkages	Formal	Scheduled Meetings between *domains* for collaboration and coordination.
Status Points	Formal	Monitoring Points and Metrics
Decision Points	Formal	Formal meetings or process points between *stakeholders* for making decisions. These could be agile in nature, such as a demonstration for user acceptance, or more waterfall based, such as decision gates in the Stage Gate process.
Touch Points	Informal	Informal interactions that occur to resolve potential problems or follow up on progress. Largely intuitive in nature.

Table 6: Understanding Interconnections and Interactions in Hybrid Agility

Dependencies are just that. They are situations where one domain has a dependency on another to complete a task. As is often the case in embedded systems, one piece of the solution, such as firmware, may have to be completed to a specific level before hardware can complete their work, or vice-versa. This is a technical dependency. In addition, respondents noted the presence of resource dependencies. Often, one domain may require expertise or consultation with another domain before it can move on. This may require a resource or subject matter expert from one domain to stop what they are working on to help out with another.

Often, the result of these dependencies is that one domain must put its work into a *sleep state* until the other domain is ready. As one architect explained:

> *"If firmware resources are diverted then the project basically is just in a sleep state until it gets resurrected."* Hardware Architect

This presents some practical problems in that once resurrection occurs, resources must be re-engaged. This may become difficult if the original participants are not available, and new resources have to be brought up to speed. Domains attempt to mitigate these dependencies and *sleep state* situations through proactive communication and coordination. Each domain communicates to others what changes they are making that could impact them. For example, if hardware is changing the way a circuit operates and firmware needs to know about it, they will communicate this to them. If hardware needs additional test modes, they will communicate those changes as well. Although this communication or *agile interaction* is often informal, the results must be coordinated in order for the domains to keep in sync. This synchronization can put limitations on iterative development. The following excerpt illustrates that although firmware utilizes development sprints, they cannot keep *developing until they are done* as is usually the case with agile Scrum methodology:

> *"And so a lot of times we have to coordinate, so the firmware team can't just say 'well, we're just going to deliver features until we run out of time.' We have to build those three features about a month before software needs them so software can do their work."* Software Development Manager

In summary, dependencies are managed through a series of informal agile interactions, as opposed to a formal process. Synchronicity between the domains is maintained by either planning ahead so that one domain does not have to wait on another, or by putting a project into a sleep state until the dependency is resolved.

Dependencies in embedded systems can be particularly complex in that there may be

multiple interdependencies intertwined between multiple domains. While dependencies can be described as *one-way* situations in which one domain is reliant on another, interdependencies consist of *two-way* dependencies between two or more domains. For example, one or more domains may be waiting on another domain, while at the same time that domain will need feedback from yet another before work can proceed. As one Architect noted:

> *"Hardware quality doesn't want to finish its final product testing until they have a final version of firmware. That may be dependent on, you know, the [software] release."* Hardware Architect

The organization mitigates these issues by using iterative development to provide enough material for dependent domains to proceed. As one manager explained:

> *"So they generally have major milestones or target dates for deliverables of features and so they'll deliver us a [device] that has 30% of the features set on it. We'll take that, we'll implement that 30%, test it, and then by the time we've done that, they've delivered the next, you know, 30% of the feature set and we'll work with them."* Software Development Manager

Interdependencies in embedded systems are essentially a complex web of intertwined dependencies that must be carefully monitored and managed to ensure that projects keep moving. To summarize, they are a form of interconnection in which two or more domains are symbiotically interdependent on each other. Such interdependencies can come in the form of shared testing and development needs, and they are often managed by one or more domains,

providing iterative functionality that allows the other domain(s) to proceed. This is one way in which process agility is managed.

Different from dependencies or interdependencies, linkages are *scheduled* interactions between stakeholders for the purpose of sharing information, coordination, collaboration, and decision making. These consist largely of a series of formally organized meetings attended by progressively smaller, yet more executive-level, teams as issues and the status move from the ground level up to C-Level. Such meetings include release architecture meetings, Scrum standup meetings (including a larger Scrum of Scrums meeting), project operations review and change control board meetings.

At the lowest (or development team) level resides the daily Scrum standup meetings. As the development organization is divided into Scrum teams, each has its own *standup* within the software and firmware domains:

> *"There are daily standups by sprint teams. Those are attended by Scrum master and/or the key people on the team. They discuss what they're working on, how they're progressing, and issues they're encountering."* Software Project Manager

As the development organization employs two-week sprints, sprint team meetings are held bi-weekly. These meetings are where requirements or user stories are reviewed with the engineers and product management to resolve issues and negotiate what the final outcome may be for a set of user stories within a sprint.

Due to the size of the organization, large distributed teams report in to small Scrum of Scrums meetings which roll into an even larger one. This is one way in which an embedded

systems organization allows the various distributed teams to *roll up together* into one Scrum. As one project manager explained:

> *"We have a Scrum of Scrums which is where we meet with all the software managers, firmware managers, and the leads and we discuss how the sprint teams are performing, and any issues that they're encountering."* Software Project Manager

Depending on the needs of the release or the project, there may be multiple *Scrum of Scrum* meetings broken up by function, as a project manager explained:

> *"There's even a smaller Scrum of Scums that meet a couple times a week and that is a little bit higher level than the standups and a little bit lower level than the project Scrum of Scrums, and those have been broken up by major functional areas."*
> Software Project Manager

At the next level (release management level) are the release architecture meetings. As a project manager explained, these meetings are attended by most first-level managers, product managers, project managers, systems engineering, and other stakeholders who may have issues on the agenda for discussion:

> *"And so we have release architecture meetings multiple times a week, which is where we review what's going on in the release. That's attended by software managers, firmware managers, and systems engineers, architects, some key experts as needed,*

and in there we review what's targeted for the scope and get things slated up for sprint work." Software Project Manager

Release architecture and Scrum meetings typically only involve software and firmware domains. Hardware is brought in at the project operations review meeting, which consists primarily of first and second-level managers in conjunction with the executive team:

"We have a project operations review every week, which is where we bubble up everything out of the project systems meeting and present that to basically everyone else in the company, the executive review board, the VPs. We give them insight into the project. We give them the opportunity to weigh in or help us with an issue or address any questions they have." Software Project Manager

As with most agile Scrum environments, retrospectives are performed to find out what could be improved upon. In a large embedded systems organization with distributed teams, this was found to be a challenge. As one development manager explained:

"Yeah, we do the retrospectives. Rolling retrospective information across 40 teams is a bigger challenge than rolling it up across three or four teams. You can't meet all together and talk about it. So in past projects where I've had three teams, you can bring 20 people

in the room and talk about a retrospective, you do it on a team-by-team basis and you can bring [roll] those results back up. Across 40 teams that a big challenge."

Software Development Manager

Another linkage type is the change control board meetings where defects or other significant changes to *released* software are discussed. The attendee list is similar to that of the release architecture meetings.

Linkages are a form of formal interconnection (or interaction) that usually consists of a set meeting or meetings that serve as formal contact points between domains. They are part of how agile processes are orchestrated across the enterprise.

Status points are monitoring points and metrics that managers use to observe progress and alert on potential problems. This activity is not limited to development but starts early, even as new requirements are decomposed and understood. In addition to the usual burn down charts, managers employ a customized dashboard that monitors progress based on requirements activity. The first of these metrics is the *decomposition rate*.

As mentioned earlier, decomposition of requirements is key to understanding them. This activity takes time, and it is important that it is monitored. The excerpts below explain how the decomposition rate is created and monitored:

"So [for] a feature that hasn't been broken down or well understood, it [dashboard] shows an estimated value of that, and we compare that to the total decomposed value and also the percent complete based on each." Software Project Manager

> *"So [in] that decomposition process we have a percentage. So I'm simplifying the math, but if we start with 10 requirements and they have, you know, between five and 20 stories each, on day one, the decomposition percentage would be zero and then as the business analysts and the product owners work and start generating stories, we'll start checking off stories that have reached a gating point."*
>
> Software Development Manager

In addition to decomposition rate, the progress of user story development and the tasks they consist of is monitored via the dashboard and a burn down chart. These burn down charts are broken down to the team level and to the individual level. These statistics can also be rolled back up to project level which shows how many *ideal engineering days* (based on approximately 6.5 work hours per day) are in each sprint and the entire release. System releases typically consist of ten such sprints.

Another important metric is velocity, which is based on how many ideal engineering days a team has completed in each sprint. Velocity performance is compared to previous releases to gain an understanding of how teams perform over time. It also serves as a benchmark for capacity and as a predictor for scoping the next release

Of course, no executive dashboard would be complete without financials and general project performance data. Budgets are tracked to the actuals of the company's financial spend. Project dependencies are monitored as well as past release metrics. Measurements of how long it took previous system releases to go from one Stage Gate to another and their respective financials are actively compared to current efforts.

Yet another important status point is defect metrics. The incoming arrival and closure rates for defects are monitored, as well as their customer impacts. All of this metric data is maintained internally in a central repository accessible by the project team.

In summary, status points are a form of interconnection that consists of monitoring points that the organization uses to keep track of what is going on with feature decomposition, development and testing. They serve as inputs to decision making and agility management.

Another type of interconnection is the decision point. These don't always occur in a meeting or specific venue and can happen throughout scoping and development. As requirements understanding is taking place, decisions are made collaboratively by the executives from engineering (process agility) and the business (market agility). These include decisions regarding what kind of work and how much can be taken on for the next systems release, as one manager described:

> *"Before we sign up for it, they're evaluating at different levels whether we're ready to take on the next "big one," and that would be when they look at their revenue plans and they see the top-line utilization of the R&D assets."* Software Product Manager

While software and firmware tend to be more agile in the way they approach decisions, hardware is much more rigid and waterfall based, requiring a feasibility study in the beginning to help decide whether, when, and how the work could be taken on. During the progression of the systems release, Stage Gates are integrated into the agile process as *check points* on the progress and reliability of the release. These check points allow all three domains to maintain

synchronicity. If the project has met its gating requirements, it will be allowed to proceed to the next gate.

Not all decision points are grounded in the Stage Gate process. Important decision points are made at the user story and requirement levels as well. The final decision point for any requirement is the demo or demonstration. Stakeholders, typically the product manager, will observe and sign-off on the demo if it meets expectations. Respondents felt as though the size and complexity of the organization contributed to a more formal demonstration process. As one manager noted:

> *"Our demo is more formal, much more formal than it has been in other companies, and I think the reason for the formality is because we have a lot of product managers, a lot of different people, and a lot of developers in place."* Software Development Manager

Such complexity contributes to limitations elsewhere, such as change management. Even in a hybrid agile environment, change becomes more rigid beyond a certain point. Although the ability to change is an important component of market agility, it does not mean that it is constant throughout the development process. As the system release progresses, it becomes less impervious to change. With embedded systems organizations in particular, the release tends to be more rigid where hardware and multiple domains are affected. The following excerpt illustrates how change is managed after the systems release has passed its Stage Gates:

> *"After that, change still happens but, you know, it's a process. It has to go through change control, it has to be well documented and with that, the team agrees that 'hey,*

this is the change we need to make, it has all the right buy in and has the right business specifications, so let's make it.'" Software Project Manager

A Hardware Engineering manager explained how such changes tend to be much more rigid in his domain:

"Before that can happen, an engineering change order has to be written that explains what's being changed, what it effects, and why it's being changed and then this ECO is routed through the various functional groups: electrical, mechanical, firmware, supply chain, manufacturing, hardware quality assurance and systems quality assurance. It communicates the change and all these functional groups have to approve that change and it also notifies them of what's changing and what the impact is on that functional group." Hardware Engineering Manager

To summarize, decision points are a form of interconnection where the Stage Gate process and agile methodology synchronize and sanity check each other. In other words, it is where the agile and waterfall sides of the organization come together, hence the management of hybrid agility.

Less formal interconnections are *touch points*. Touch points are informal interactions performed by managers and other stakeholders to *check on* what may be going on in another domain or team. It is a form of tacit communication that is always going on, yet it is not formally required or stated. The initiating of such communication is largely intuitive, but has proven effective in making sure tasks are being performed, roadblocks are removed, and that

processes are being orchestrated as expected. These touch points can be one-off communications for follow up or ad-hoc meetings to resolve issues or to continue requirements decomposition. One manager described this as:

> *"Helping [to] ensure that the teams are completing what they need to complete, when they need to complete it."* Software Project Manager

Systems engineering plays a significant role in managing these interactions, along with project management. They ensure that business requirements are properly broken down into technical requirements for each domain, and serve as the primary communication conduit from the engineering organization up to the business:

> *"I will interact with systems engineering and systems might go to the change meetings. Systems would also act as the go-between between product management and firmware."* Firmware Architect

These communications occur at all levels of management, as one project manager explained:

> *"Then I work with product managers on a regular basis, the directors and the VPs to assess the project, determine where we are, how we need to proceed, let me know if there are issues with scope or some new customer commitment. I meet with them, kind of on a regular, not a scheduled basis but a regular basis."* Software Project Manager

One respondent stated that documentation can sometimes take the place of interpersonal interaction as a touch point:

> *"So usually the way that those touch points happen would be us developing a technical specification."* Hardware Architect

Touch points are a form of interconnection that consists of ad-hoc meetings, documentation, and personal follow-up. It is a largely intuitive part of the process because it may be initiated by the project manager or other stakeholder based on feel, discomfort, or output from a monitoring tool that lets them know they need to initiate a meeting or contact a stakeholder for status.

As the information inputs from the various interconnections and interactions are realized, the company makes adjustments. Promises are made intuitively and quickly with little information and are actively balanced with contractual workload. Adjustments to scope, resources, and customer acceptance in particular are an important component of agility management. These adjustments are updated via a process of constant reassessment.

Customer collaboration is a key tenet of the *Agile Manifesto*. Respondents indicated that much of their work involved influencing customer acceptance of the product. By working with the customer to develop different modes of acceptance, products could be brought to market quicker. These modes most often consisted of field trials and pilot projects. Field trials are where the customer receives an early version of the product and is allowed to test them and provide feedback. With this technique, the customer benefits by getting a new product quicker

and having the chance to influence the product direction, while the vendor company saves resource costs by essentially outsourcing its testing to the customer, as a hardware product manager explained:

> *"And that's the first chance for us to get some feedback on our quality. Our customers in Canada, they do really, really thorough testing of our products. I would say sometimes even more detailed testing than us, so we take those feedbacks, and that helps us to improve if there is an improvement needed in the quality of [our] tests, that's certainly a good thing."* Hardware Product Manager

Pilot projects are another method of agile customer collaboration. Using this method, the customer's expectations on quality are lowered in exchange for the opportunity to *be first*. This allows the vendor company to bridge customer needs with organizational capabilities, as a software product manager explained:

> *"We work with that customer to set expectations that we are going to pilot things with them instead of giving them a proven, field-ready, tried-and-true product, and the customers, to their credit, have generally accepted some of these decisions and worked with us as long as the expectations were managed."* Software Product Manager

Manipulating customer acceptance is one way in which the organization makes adjustments to manage agility. Through the use of field trials, pilot projects, and other modes of

acceptance, the organization influences as well as adapts to the business momentum generated by market pressures.

As with managing customer acceptance, adjusting scope is one of the necessary evils of managing agility. Due to the high unknowns of new technology and changing customer needs, capacity is pushed to its limits and is often over-estimated, then it is gradually adjusted as the requirements and business needs become more apparent. This refinement occurs gradually as requirements are better understood. Often, this may continue after decomposition and well into development.

One respondent noted how they over estimate capacity or *pack the release* with the expectation that items will be pulled later:

> *"I get a lot more of 'well, I want you to prioritize three times the capacity of the project because I really don't know which bits and pieces I'm going to pull to be able to fill up the actual capacity.'"* Operations Manager

Another respondent recounted how requirements are selected for the release as they bubble up to the top:

> *"We pick the highest priority items off of the top of the pile and slate those to a release, haggling over what's really a priority and so forth [until it] is finally settled."* Software Product Manager

These scope adjustments are often strategic. They may be based on obtaining business from a specific customer or sector or be due to the lack of profit in a specific product line.

Respondents noted that revenue generation tended to be a key component of the company's strategic direction. In order for rapid scope adjustment to work, the organization must be flexible in its ability to abort gracefully on requirements, features, and/or products. These requirements may be postponed, or dropped altogether. Resources can then move quickly from one aborted task to a more important priority. Such decisions are made at the executive level, with the business or product side working in tandem with the engineering or organizational side to make the ultimate decision.

Resources must be adjusted, as well as scope. Analysis revealed that the organization cultivated an ability to flex resources in a variety of ways. These consisted of maintaining team flexibility, outsourcing when needed, and most importantly, relying on a core group of engineers with high expertise. Such flexibility is much higher within the software domain than the firmware or hardware domains, but it still exists. The reason for this difference was cited by many respondents as being due to the lack of interchangeability of resources. Such interchangeability is less prevalent in the firmware and hardware domains due to the specialized level of expertise required.

Teams have the ability to optimize the use of this high expertise when necessary. The agile concept of self-organizing teams and pair programming allows them to organize the required expertise according to the current scope. Although expertise may be high and specialized, respondents noted that the teams are smart enough to organize the *right mix* of people. Having these self-organizing agile teams was found to be critical in maintaining capacity, as one manager explained:

> *"If we don't have agile teams, if we are constantly swapping in new features, if Team A only works with one type of code or one type of functionality and that feature is now pulled from the release, well now that team goes unused and they have to scramble to do something else or we're going to lose capacity."* Software Project Manager

Embedded systems development brings with it its own set of challenges with regards to high expertise and self-organizing teams. As mentioned previously, firmware sits in the middle of the technological solution between software and hardware. Resources from firmware are often strained because the other domains require their support. Managing this *resource rotation* is a continual challenge. These resources tend to be even more specialized and less interchangeable than other domains, as a manager explained:

> *"We have firmware guys that are rotating in and out, say for instance 80% of the time they're supporting the software group and 20% of the time they're supporting hardware. If I've got a firmware guy that's supporting hardware efforts and he gets moved over halfway through the life of development to support software and we bring somebody else in that knows nothing about this hardware development it's a challenge for him to get up to speed."* Hardware Engineering Manager

The most severe example of this flexibility is referred to as the *hero model*. Often as a last resort, the organization will draft one, or more, highly capable expert to solve a problem or meet the goals of a release entirely outside of the agile processes, as a lead architect explained:

> *"The hero model, which we know doesn't really last forever, it's not a good thing to build a company on, but sometimes when you've got to get something done really quickly and you don't have time to track story points and break it down, you can just give it to a group of very capable people and say, here, you just need to get this done as quick as possible."* Software Architect

The organization must continually adjust the correct resource mix across the range of domains and projects. This adjusting is facilitated by a constant process of reassessment of the business's current position against its strategic direction. In this way, the business reassesses all of its adjustments.

Respondents noted that much of this reassessment activity arose from the hybrid agile implementation. The Stage Gate method forces re-evaluation at each gate that many felt makes the organization more agile, despite its waterfall nature, as a manager explained:

> *"Because it's within that waterfall process, it probably makes us more agile because we have to constantly reassess and reevaluate where we are and what we need to complete versus just finishing what we finish."* Software Project Manager

Agile orchestration is the group of activities used to manage agility across the enterprise. It is how process and market agility are managed to achieve a common goal. There are two major categories of agile orchestration, which are interconnections or interactions, and making adjustments. Interconnections consist of dependencies of one domain on another, interdependencies between two or more domains, formal linkages or key meeting points between

domains, status points for monitoring and maintaining status, decision points for executive decision making, and informal touch points that stakeholders establish intuitively to keep the process moving. Decision points and linkages also serve as the connecting points between the agile method process and the waterfall process, and they therefore assist in managing the hybrid agility implementation that the organization has employed. It is important to note that the kinds of interconnections and interactions developed within this study are largely influenced by the embedded systems context. The "agile characteristics" outlined in figure 2 enable all of the domains to work together as one cohesive unit, agile orchestration ensures that this cohesion occurs. The interactions and interconnections are designed to bring all domains within embedded systems together both informally and formally when necessary to ensure the production of the systems release. The necessity of this cohesion and the agile characteristics and orchestration it demands are specific to embedded systems.

The business then uses the outputs and inputs from these interconnections to make adjustments to customer acceptance modes, scope, and resources to manage the agility of the organization. The process of making these adjustments is one of continual reassessment.

Agile Vortices: The Grounded Theory

Through open, axial, and finally, selective coding, grounded theory methodology maintains that a central theory should be identified. Strauss and Corbin define this phenomenon as the central problem that the subjects are trying to solve. Strauss and Corbin further hold that other categories should be explained in terms of this central theory (Strauss, 1990). The previous sections illustrated the primary categories identified via axial and open coding and the elements that compose them. These include market agility, process agility, business momentum, and the systems release. In this section, we explain these categories in terms of the central

theory. Using concepts from fluid dynamics, combined with the metaphor of a whirlpool, a succinct visualization is provided which describes how all of the categories are linked together into one comprehensive model.

Figure 3 below combines the two figures previously mentioned, Figure 1, and Figure 2, into one view. It illustrates how the hybrid agile organization of software, firmware, and hardware combine with the product genesis of the business as a result of market pressure.

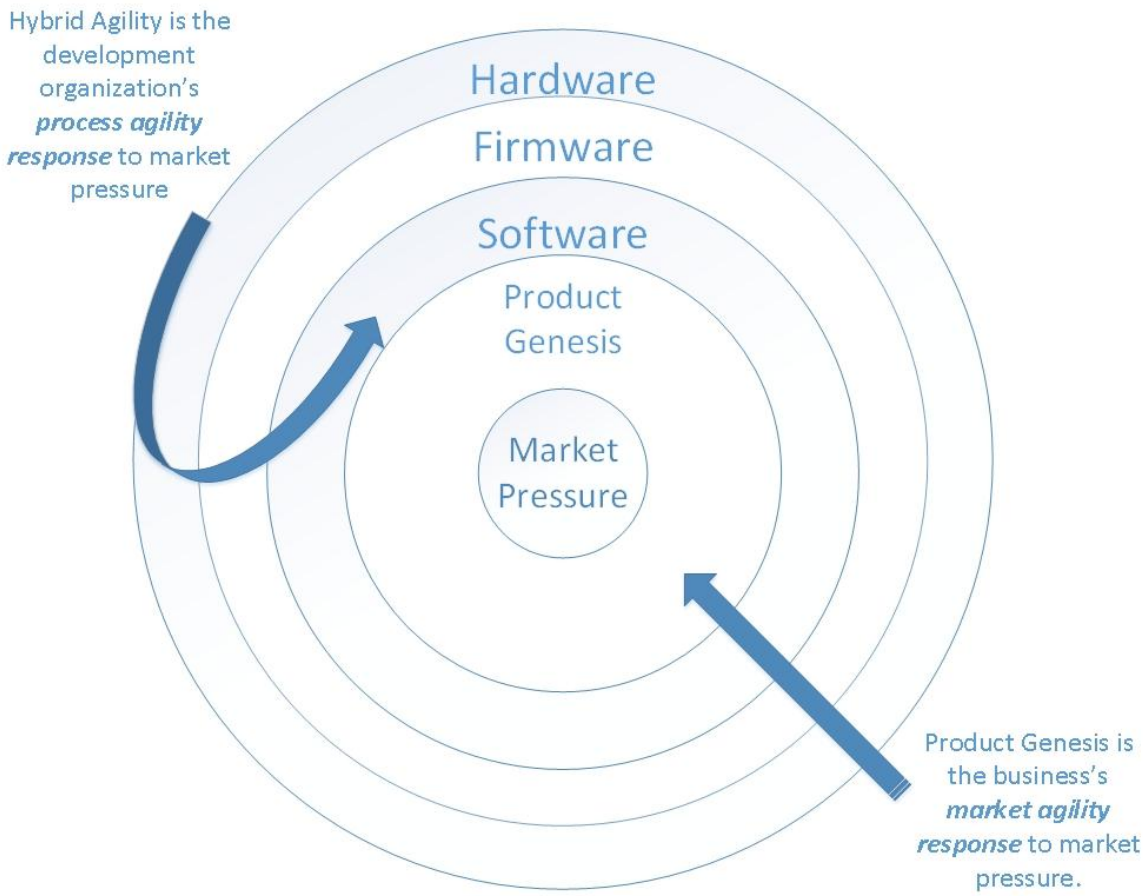


Figure 3: Rolling up Process and Market Agility Categories into One View

During selective coding, an analysis of the data indicated that both sides of the business, the product management organization and development, are constantly attempting to reach the

same *point* throughout each product release and will manage themselves into making this happen. According to the software project manager:

> *"Usually we determine when the release is going to go out the door and then from there we back into how much development can we squeeze in, and we really say how much quality are we willing to accept within this period and if it works out then that period stays. If we need more quality then we'll reduce capacity of the release and do less development."* Software Project Manager

This point of convergence was identified as the central theory . The reason it is identified as such is because it is the central problem that the subjects are trying to solve. Essentially it is the *gravity* that pulls all of the categories identified in axial coding together. Figure 4 below illustrates this point of convergence. Product genesis is the business's *market agility* response to market pressure. Product genesis in turn *sets the tone* through its creation of business momentum. The development organization attempts to match this momentum through the creation of the systems release, which is created by the hybrid agile development organization. Hybrid agility is the development organization's *process agility* response to market pressures.

Figure 4: Business Momentum and the Systems Release are created by Market and Process Agility

These linkages can best be explained using a metaphorical illustration, as part of the selective coding process.

Consider a whirlpool as a metaphor for the subject of our study. Whirlpools are a form of vortex, which spin around a central axis. Based on fluid dynamics, the velocity of the rotation in a whirlpool is greater as you get closer to this axis. Suddenly, a tennis ball falls into the pool. As the ball is drawn closer to the axis, it acquires a spin or rotation of its own and moves at a velocity and direction influenced by the vortex. As it does so, it gains *momentum,* based on its mass or size multiplied by its velocity. The movement of this ball illustrates the motion or circulation of the vortex. The circulation of the vortex at the position of the ball is its *vorticity*. Vorticity has been defined in fluid dynamics as the point in a vortex where the *curl is the strongest*. One firmware manager characterized how momentum is felt within his organization:

> *"It (change) kicks off a whole chain of events that goes on, so I think there's always a lot of momentum going with project schedules. There's a lot of momentum going. If you have something that changes midstream within a project then it's very hard for us to change direction there, and it's got to be kind of planned into future releases."*
>
> Firmware Manager

Using this metaphor, we can easily map Figures 1 through to 4 to the whirlpool. The central axis of the whirlpool illustrates the effect of market pressure. The innermost ring of the pool is product development or product genesis as we described earlier. This ring consists of requirements development based on customer input, as influenced by market pressure, and it is led largely by product management in conjunction with systems engineering. As the innermost ring, it spins the fastest. The next innermost ring is the software development part of the

organization. Software development can occur independently or in conjunction with other domains such as firmware and hardware. Because it runs on a fast release cycle of six months or less, it is the next innermost ring. Firmware and Hardware domains make up the next two rings respectively, with hardware furthest to the outside. Firmware is often managed within a software context but has linkages to both software and hardware within the organization. Hardware makes up the outermost ring because it operates on the slowest release cycle of all, which can be up to two to three years. Although all three domains can and do operate and release independently, during a full system release they must all be in complete alignment. This is a unique property of embedded systems and illustrates one of the key challenges present in this context. As in a whirlpool, although each ring is interconnected they are all running independently at progressively slower velocities as the observer looks outward from inside the vortex at the observation point of the tennis ball (refer to Figure 5). These domains and the management of them constitute our hybrid agility implementation.

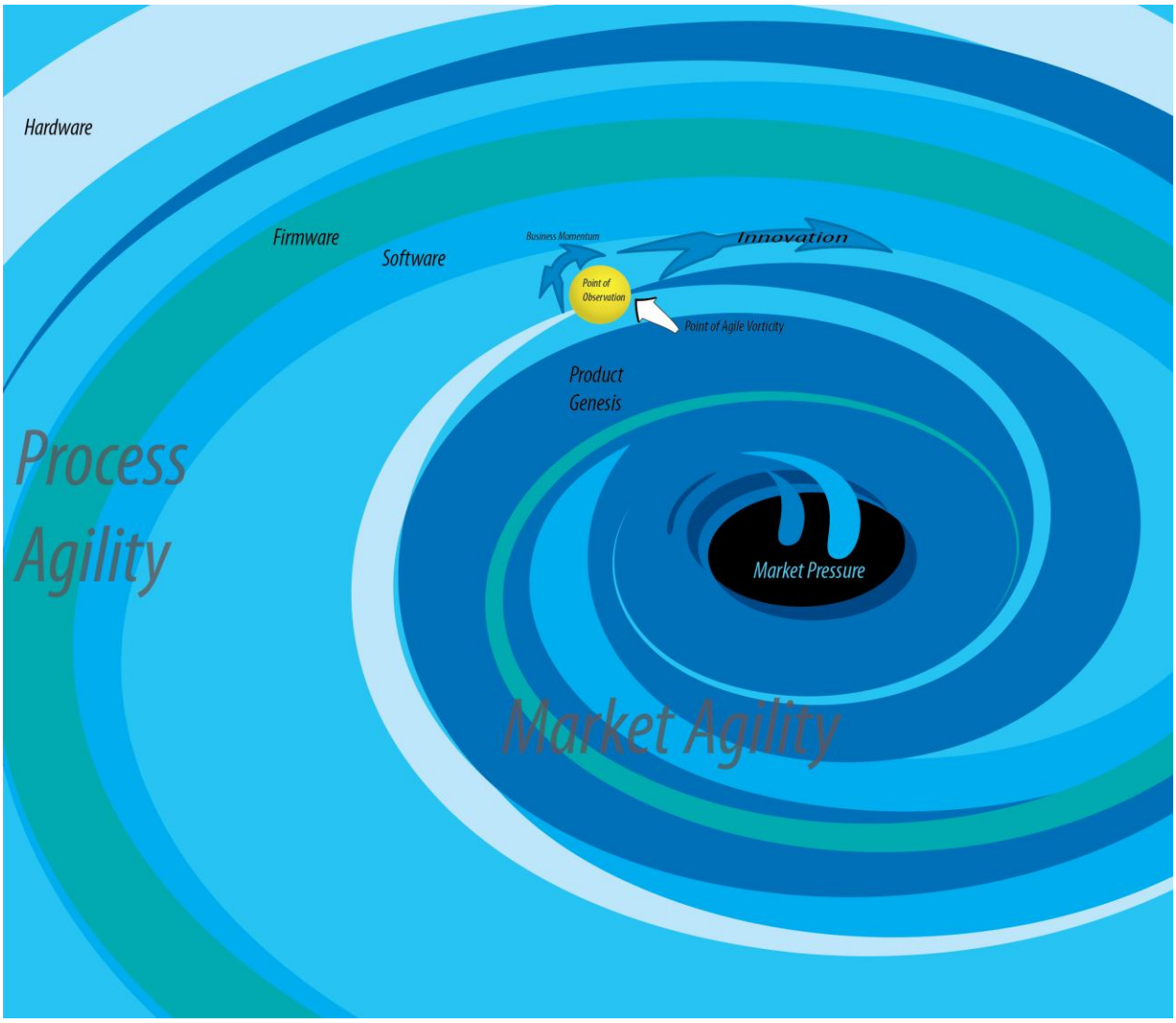


Figure 5: The Agile-Business Vortex: The Ultimate Goal of Agile Orchestration is the Management of Process and Market Agility to achieve Agile Vorticity

The tennis ball in our metaphor falls between the first ring, *product genesis*, and the second ring, *software*. The position here represents the dividing line between *market agility* (the area between market pressure, product genesis and the ball) and *process agility* (consisting of software, firmware, and hardware). Market agility is the ability of the *business* to adapt to change in the market and is a function of product genesis. Process agility is the ability of the *organization*, including software, firmware, and hardware, to adapt accordingly through hybrid agility.

The vortex could include a mass, or size, which can represent the scope of a specific system release or a series of system releases over time. This can include a *product scope* or roadmap. The velocity of the ball is the *timeline* at which this system release or product roadmap is to be achieved. Multiplying the scope size by the timeline velocity produces *business momentum*. The *direction* that the ball is moving illustrates the technical direction of the product roadmap, or *innovation. (Refer to Figure 2.)*

Momentum = Mass X Velocity

Business Momentum = Scope X Timeline

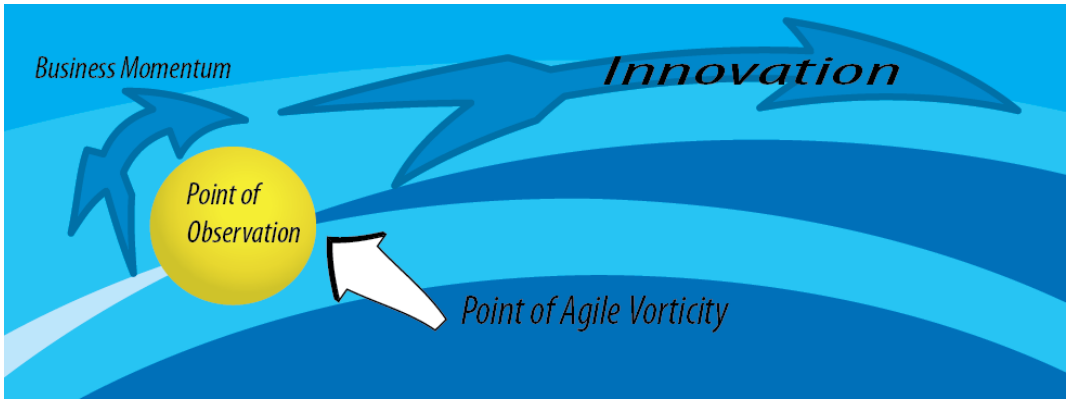


Figure 6: Agile Orchestration Close-up: Business momentum, Innovation, and Agile Vorticity

Finally, the circulation of the water at the point where the ball is located is called its *vorticity*. This is the point at which everything converges. *Market* agility is the ability of the business to reach the vorticity point with its product roadmap under the influence of market pressure via product genesis. *Process* agility is the ability of the organization through hybrid agility to reach the same point of vorticity. A good illustration of process agility in this illustration is an outstretched arm attempting to reach across the organizational rings to reach the

ball, as market agility slowly sucks it further away. It should be noted that time in this metaphor is ever present, as it would be in actuality. Vorticity is relative to the point of view of an observer at the same point of observation, moving along with the fluid.

Agile orchestration is the creation, nurturing, and closing of an agile business vortex in which market and process agility intertwine to produce a new software release. This was found to be the central problem that all aspects of the organization were trying to solve. These agile business vortices which are created as a result of high market pressure in conjunction with high technological innovation are the central theory of this study. The aforementioned model depicts bringing multiple forces together that create a need to be agile. Each concentric ring influences the point of Vorticity where the firm needs to be to successfully produce a systems release.

Chapter 7: Discussion

The purpose of this study was to determine how agile processes are orchestrated in an embedded systems context. The result was an empirical analysis of a hybrid agile implementation involving high innovation within a turbulent marketplace. We begin this section with a discussion of this hybrid agile implementation, how it is managed and the forces that created it. This discussion is followed by an exploration of *fluidity* and how this concept links together hybrid agility, embedded systems, continuous releases, and innovation, within the context of our *fluid* whirlpool metaphor.

An Inquiry into Hybrid Agility

As our vortex metaphor implies, a hybrid agile implementation is a complex one, subject to powerful forces of market and innovation, thereby making the management of it particularly challenging. So how does the organization in such an environment organically adapt to these forces, and can they actually be controlled? Based on the results of our study, hybrid agility is a delicate balance of agile methodologies and Stage Gate processes. While the agile aspects of this balance allow for higher degrees of market response, the Stage Gate characteristics function largely as the *boundary conditions*. They serve as the check and balance against agility. This is due in large part to the embedded systems context and the constraints that such technology places on an engineering firm. Embedded systems environments include not one, but multiple development domains that operate independently yet are forever linked. While the business as a whole considers itself agile, each domain within the embedded context has adopted agility in very different ways.

Software, the most nimble of the domains, has adopted agile Scrum methods almost entirely. As a result of this high level of adoption, they serve as the *early responders* of the

engineering team. By contrast, the slowest of the domains, hardware, has not adopted any agile methods at all and remains largely Stage Gate managed. Despite this fact though, our study found that hardware does employ lean concepts of rapid prototyping and a *fast track* Stage Gate pathway that it uses to maintain rhythm with the rest of the company. In the middle is the firmware domain, which has employed some aspects of agile and Scrum in terms of requirements management and standup meetings, yet stays away from the rigidity of two-week development sprints. Due to the shared resource nature of firmware, its complexity and the specialized expertise required to develop it, breaking up work into small, rigid iterative sprints is not very feasible.

In this way, the nature of the different domains places boundaries on the level of agility each can accept. Additionally, as hardware is the slowest domain and the primary profit center for the company, the Stage Gate process used to manage it also used to keep the other domains grounded. Regardless of their level of agile adoption, stakeholders from each of the three domains must check in at the various gates within this waterfall process. In this way, the boundary conditions of hybrid agility are largely provided for by this Stage Gate process.

Why has the engineering organization in our study adopted agility in this way? As explored earlier, causal factors for organizing development in ways such as this have been found to be:

- a desperation to rush market,
- a new and unique market environment,
- a lack of experience developing under the conditions imposed by the environment (Baskerville et al., 2003; Lyytinen & Rose, 2005).

As mentioned previously, the nature of the smart grid technology and the power utility market have created a *gold rush* situation. This is definitely in line with the first two causal

factors. Secondly, although some components of the business being studied have been around for years, the current combination of merged organizations has only been in place for a relatively short time. Adoption of agile methodologies within the business was started only a few years ago. Such adoption occurred fluidly and organically over time, because no one involved had much prior experience implementing agile in a complex embedded systems environment with such high market turbulence.

Fluidity and Continuous Releases

The implementation and orchestration of hybrid agility can be at least partially explained by a fluid view of agile methodology. Allowing agile implementations to be tailored provides for better accommodation of change, especially when frequent releases are necessary (Baskerville et al., 2003; Lyytinen & Rose, 2005). This can be further enhanced with parallel development which allows developers to correct problems as they occur. As with the different domains within embedded systems, it has been shown that different methodologies can be isolated for different releases (Baskerville et al., 2003). Further, this fluid view of development methodology provides a framework that can contain the behavior of system components that have been developed with different approaches, such as software developed with agile and hardware, created with waterfall.

Methodological flexibility allows different teams to find their ideal working style given the mix of the group, such as firmware teams versus software teams (Baskerville et al., 2003). It also allows developers to vary their approaches when environmental constraints change, such as the examples of *C-Level* and *hero model* approaches in our study (Vidgen, 2009). All of these *fluid methodology* characteristics are in line with our findings of hybrid agility. Although the literature shows that boundaries are needed on process innovation, we can see these boundaries

in our study with the adoption level of each embedded domain and the decision points provided by stage gating.

This fluid approach to process innovation is likely to continue to influence the subject of our study as well as the industry at large. Recent studies have noted a movement from agile methods to more lean practices in software development (Wang, Conboy, & Cawley, 2012). Kanban is a good example (Sjøberg et al., 2012). When one examines the agile business vortex, it is easy to see that as business momentum increases and the point of vorticity becomes more challenging to achieve, the organization may be required to move from the time-boxed iteration style of Scrum to the more *fluid* process of Kanban. This strategy combines both event and time pacing into more of a *flow*. Such a strategy can better accommodate more continuous releases with less lead time (Sjøberg et al., 2012). Indeed, in some ways the subject of our study has already expressed some tendencies towards this end. Even though time-boxed iterations are used within the company's agile process, *event pacing* is employed when necessary with such techniques as the aforementioned *hero model*. This allows the development organization to get things done *on the fly*, thereby allowing the business to be more reactionary when needed.

Hybrid Agile Implementations: Whirlpools within a "River of Innovation"

Innovation has also been characterized as a sort of *flow* (Rogers, 2003). Innovation takes place when a technology is created, and more innovation occurs as that technology is transferred to others (Rogers, 2003). In other words, when one event happens upstream it triggers other events downstream, just like a river. These events can be influenced by market dynamics and technology turbulence. With respect to our agile business vortex, agile is accelerating the response to increasing market pressures which in turn is creating these whirlpools within a river of innovation. This increased agile response, and the resulting whirlpool, place higher demands

on the organization. As implied earlier, this demand may force an organization to supersede the time-boxed agile iteration with a *Kanban* type of flow just to keep up.

In the latest version of his book, *Diffusion of Innovations*, Rogers notes the following research opportunities with respect to innovation development processes (Rogers, 2003):

- How are user's needs and problems communicated to development teams?
- To what extent are technological innovations developed by *lead* users instead of research and development experts? Is the creation of innovations by end users a general pattern?
- What are the key linkages and interrelationships among the various organizations involved in the innovation development process?

In the context of embedded systems development and hybrid agility, this study provides answers to these questions. It shows how user's needs are communicated in a hybrid agile environment. This process begins with product genesis, the continuous activity of requirements comprehension and refinement. Expectations with customers are then actively managed and negotiated by the engineering organization as the product is iteratively developed. Finally, different modes of acceptance are negotiated with the customers, which typically include intense customer involvement in the testing process.

Customers willing to accept a *less than perfect* product in exchange for added influence in product direction, enhanced service levels, and the chance to be an *early adopter* could well be considered *lead users*, as Rogers describes them. When it comes to highly innovative products or technologies, requirements comprehension within product genesis can only get so far due to gaps in knowledge. *Lead users*, in the context of hybrid agile embedded systems, are critical to bridging this gap. This gap bridging is an element of *customer acceptance* within agile

orchestration. It is one way in which the designated point of agile vorticity is reached. To further answer Rogers' query, it is indeed a general pattern with respect to our context.

Finally, the results of the study explicate in detail the linkages and interrelationships among the embedded systems development organization (including software, firmware, and hardware domains), the business, and how these are *orchestrated*.

Chapter 8: Conclusion

There has been a noticeable proliferation of hybrid agile solutions which have evoked interest from both research and practice alike. The objective of this study was to determine how agile methods are orchestrated in such an important context, with the added complication of embedded systems development. To perform this study, key informants were interviewed with direct responsibility of managing agility and related processes across the enterprise. This was further enriched with informants from each embedded domain, including software, firmware, and hardware development. What resulted were new learnings with regards to hybrid agility, embedded systems, and process innovation.

Implications for Research

Our study discovered that hybrid agility can include a mix of agile, Stage Gate, and even lean concepts, depending on the domain, project, and development context. The optimum mix for this hybrid approach is often actively tailored to the needs of the organization. Additionally, our theory of agility orchestration in the vortex of embedded systems provides a deeper understanding of how hybrid agile is adopted in embedded systems, how it is managed, and the enablers or inhibitors specific to this context. Most importantly, our inquiry into the orchestration of agility revealed new insights on some very interesting processes and behaviors, such as product genesis, customer appetite, business momentum, and agile vorticity. As there are not many studies involving agility in embedded systems development, or in combining agile with Stage Gate processes, we believe our study is an important addition to both of these branches of research.

One of the primary drivers for adopting agile methodologies (and indeed, a key tenet of the *Agile Manifesto*) has been stated as the need for a higher level of customer responsiveness

(Alliance, 2001). Our research shows that in particularly turbulent markets with high technical innovation, whirlpools or *agile business vortices* can result. Agile innovation creates the whirlpools due to its high responsiveness to market demands or pressures. Despite the existence of such whirlpools, these forces do not run amok. We found that the organization uses agility to manipulate as well as respond. Product genesis combined with different modes of customer acceptance, and customer appetite for innovation all place limitations on *how high the vortex can be revved*. Interestingly, the literature of agile methodologies is relatively silent with respect to such limitations.

Beyond customer responsiveness and technical innovation, the delineation of a clear goal or *end game* with respect to agility is also seemingly absent in the literature. The subject of our study was found to actively seek out a *sweet spot* that it can *back itself in to* when it needs to conduct an enterprise-wide systems release. Doing so required the creation of some very elegant techniques for project management, systems engineering, and customer management across the enterprise. How this *agile vorticity* occurs in embedded systems is particularly important because of the different levels of agile and Stage Gate integration in each domain.

In addition to these learnings in hybrid agility and embedded systems, our work contributes to agile process innovation as well. The current state of agile methodology literature has been said to be in a largely *post agile* mode where the chief concerns have shifted from agile versus plan driven and workflow, to simply creating agility in a variety of ways in all aspects of development (Baskerville et al., 2011). This process innovation of agility is focused on proactively creating fast responses to changing requirements and frequent releases using concepts from other methods such as Stage Gate and Kanban. Our study shows that this process innovation was impacted by the desire to reach a point of agile vorticity, a desire shared by

release development and product management. The results of our research show that lean methods of rapid prototyping and *event pacing* or *hero models* were often used in place of time-boxed iterations. Elements from Stage Gate models were used as *decision points* or boundaries against pure agile implementations. These are all examples of process innovation. Although these boundaries were largely influenced by the various embedded systems domains, the desire to reach a point of agile vorticity was the driving factor. This same desire for agile vorticity also impacted requirements comprehension and the linkages and interrelationships used to manage the hybrid process. Using these interconnections, *lead users* (Rogers, 2003) were employed extensively to bridge the gap between product knowledge within the organization and innovation. Out of all of this activity, the central theory of agile vortices proved to be the common denominator.

Implications for Practice

In industry, agile methods are seldom seen in clean form. A practical implication of our study is that it shows in detail one framework for combining agile and Stage Gate methods. There is not likely to be a one size fits all solution for building such a hybrid approach. As our research implies, process innovation is tailored to its respective environments. Each organization must focus on its own development context, projects, and limitations. In developing an approach to process innovation, the concepts of agile and Stage Gate, and what these methods bring to the table should always be considered. The framework brought forth in this study could be used as a *playbook* for similar organizations to manage a hybrid approach of their own. In addition, the study could provide beneficial directions for exploration. How to effectively tailor these strategies to different contexts is yet to be explored and is worth studying.

Practical recommendations could include the introduction of more lean methods into the current hybrid mix. A move from iterative agile development to methods such as Kanban may reduce the amount of work in progress and allow for better process flow between embedded domains. This move would be in line with other research findings, as more organizations with mature agile adoptions are beginning to move in this direction (Wang, Conboy, & Cawley, 2012).

Kanban has been shown to be well suited to situations where great uncertainty and high amounts of change occur more frequently than that allowed by agile iterations (Wang, Conboy, & Cawley, 2012). The use of *hero models* and C-Level projects may indicate that the subject of our study is experiencing such conditions. The literature has explicated that development teams will often resort to such methods if the existing process seems to be falling short (Vidgen, 2009). This organization has also been working with a hybrid agile environment for a few years now and the current implementation is considered relatively mature. Based on the literature, this indicates that embedded systems organizations may consider moving to leaner methods. (Wang, Conboy, & Cawley, 2012). Another indicator for a need to move to leaner methods could be difficulty or failure to achieve a point of agile vorticity. Very high responsiveness to market pressures can continue to increase to a level that demands replacing the time-boxed agile iteration with more of a Kanban flow. Organizations considered mature in their adoption of agile or hybrid approaches should be mindful of their agile vorticity. This may indicate that it is time to change the approach to continuous process innovation in their business.

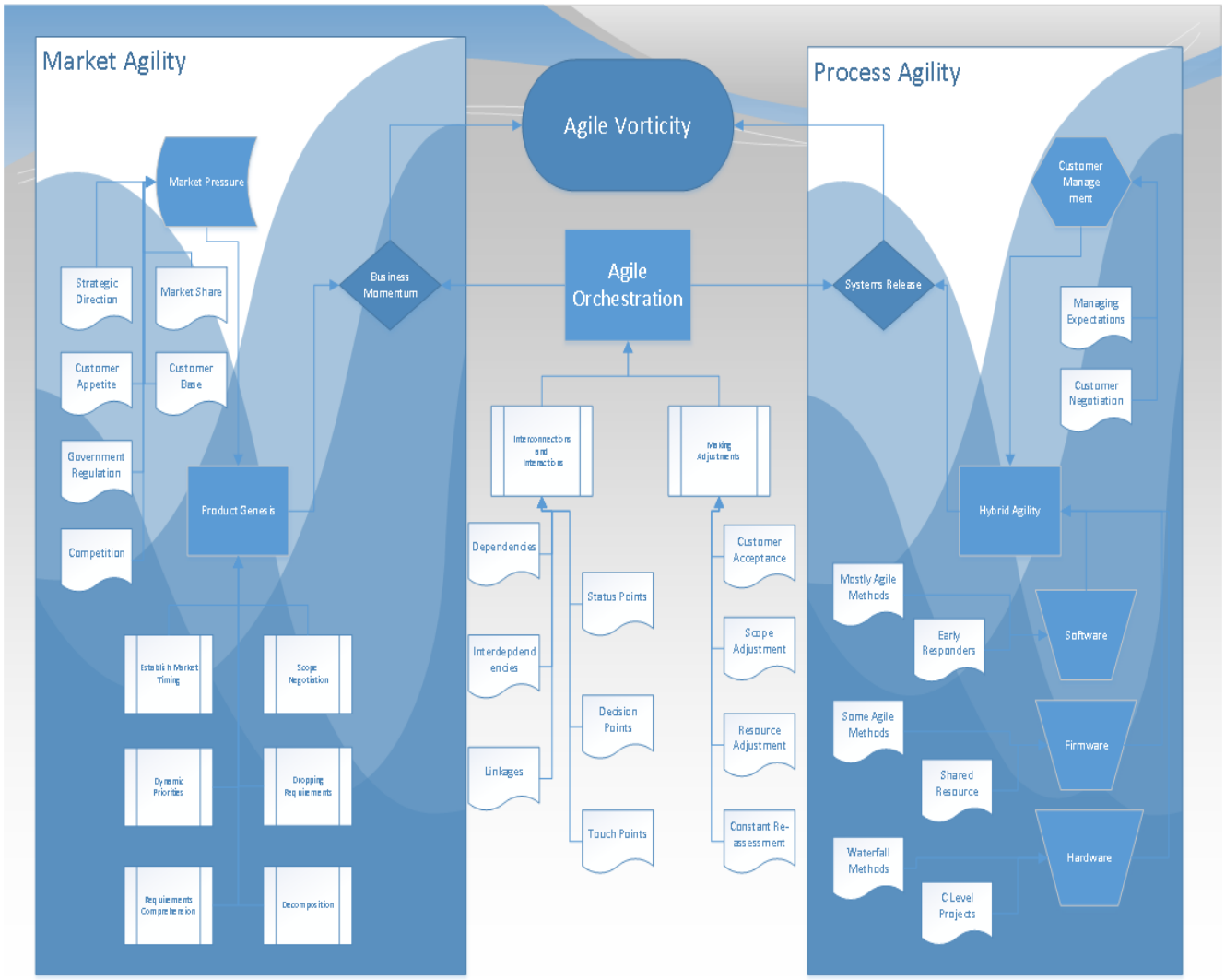
Limitations and Opportunities for Future Research

A limitation on this study is the fact that it was conducted with one case. It was also conducted in an embedded systems organization. Future studies could expand on this research by applying it to a larger number of organizations and a wider variety of development contexts.

Another future research opportunity could be a longitudinal study on how a hybrid agile implementation is organically built over time. Determining how interconnections or adjustments are established as agile methodologies are slowly integrated into existing Stage Gate environments could provide new insights. As people interactions are a key tenet of the *Agile Manifesto*, research on understanding how these interactions are established and routinized, perhaps intuitively, could also be promising. The outputs from such studies could provide new frameworks for agile orchestration and new ways to achieve agile vorticity.

Appendix

Figure 7: Agile Business Category Diagram


Market Agility
Market Pressure
Strategic Direction
Market Share
Customer Appetite
Customer Base
Government Regulation
Competition
Product Genesis
Establish Market Timing
Scope Negotiation
Dynamic Priorities
Dropping Requirements
Requirements Comprehension
Decomposition
Agile Vorticity
Business Momentum
Agile Orchestration
Interconnections and Interactions
Making Adjustments
Dependencies
Interdependencies
Linkages
Status Points
Decision Points
Touch Points
Customer Acceptance
Scope Adjustment
Resource Adjustment
Constant Re-assessment
Process Agility
Systems Release
Customer Management
Managing Expectations
Customer Negotiation
Hybrid Agility
Mostly Agile Methods
Early Responders
Some Agile Methods
Shared Resource
Waterfall Methods
C Level Projects
Software
Firmware
Hardware

References

Abrahamsson, Pekka, Conboy, Kieran, & Xiaofeng, Wang. (2009). 'Lots done, more to do': the current state of agile systems development research, Editorial, *European Journal of Information Systems*, pp. 281-284. Retrieved from http://ezproxy.gsu.edu:2048/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=44385902&site=bsi-live

Alliance, The Agile. (2001). Manifesto for Agile Software Development. Retrieved August 12th 2014, 2014, from http://www.agilemanifesto.org

Arteta, B. M., & Giachetti, R. E. (2004). A measure of agility as the complexity of the enterprise system. *Robotics and Computer-Integrated Manufacturing, 20*(6), 495-503. doi: http://dx.doi.org/10.1016/j.rcim.2004.05.008

Barlow, Jordan B., Keith, Mark Jeffrey, Wilson, David W., Schuetzler, Ryan M., Lowry, Paul Benjamin, Vance, Anthony, & Giboney, Justin Scott. (2011). Overview and Guidance on Agile Development in Large Organizations. *Communications of AIS, 29*, 25-44.

Baskerville, Richard, & Pries-Heje, Jan. (2004). Short cycle time systems development. *Information Systems Journal, 14*(3), 237-264. doi: 10.1111/j.1365-2575.2004.00171.x

Baskerville, Richard, Pries-Heje, Jan, & Madsen, Sabine. (2011). Post-agility: What follows a decade of agility? *Information & Software Technology, 53*(5), 543-555. doi: 10.1016/j.infsof.2010.10.010

Baskerville, Richard, Ramesh, Balasubramaniam, Levina, Linda, Pries-Heje, Jan, & Slaughter, Sandra. (2003). Is Internet-Speed Software Development Different? *IEEE Software, 20*(6), 70-77.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69. doi: 10.1109/2.976920

Boehm, Barry, & Turner, Richard. (2005). Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *IEEE Software, 22*(5), 30-39.

Broadus, William. (2013). The Challenges of Being Agile in DOD. *Defense AT&L, 42*(1), 4-9.

Cao, Lan, Mohan, Kannan, Peng, Xu, & Ramesh, Balasubramaniam. (2009). A framework for adapting agile development methodologies. *European Journal of Information Systems, 18*(4), 332-343. doi: 10.1057/ejis.2009.26

Conboy, Kieran. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research, 20*(3), 329-354.

Conboy, Kieran, Coyle, Sharon, Xiaofeng, Wang, & Pikkarainen, Minna. (2011). People over Process: Key Challenges in Agile Development. *IEEE Software, 28*(4), 48-57. doi: 10.1109/MS.2010.132

Conboy, Kieran, & Morgan, Lorraine. (2011). Beyond the customer: Opening the agile systems development process. *Information & Software Technology, 53*(5), 535-542. doi: 10.1016/j.infsof.2010.10.007

Dingsøyr, Torgeir, Nerur, Sridhar, Balijepally, VenuGopal, & Moe, Nils Brede. (2012). A decade of agile methodologies: Towards explaining agile software development, Editorial, *Journal of Systems & Software*, pp. 1213-1221. Retrieved from http://ezproxy.gsu.edu:2048/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=74095428&site=bsi-live

Douglass, Bruce Powel. (2004). *Real Time Agility*. Upper Saddle River, NJ: Addison-Wesley.

Drury, Meghann, Conboy, Kieran, & Power, Ken. (2012). Obstacles to decision making in Agile software development teams. *Journal of Systems & Software, 85*(6), 1239-1254. doi: 10.1016/j.jss.2012.01.058

Dybå, Tore, & Dingsøyr, Torgeir. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9–10), 833-859. doi: http://dx.doi.org/10.1016/j.infsof.2008.01.006

Eisenhardt, Kathleen M., & Brown, Shona L. (1998). TIME PACING: COMPETING IN MARKETS THAT WON'T STAND STILL. (cover story). *Harvard Business Review, 76*(2), 59-69.

Fitzgerald, Brian, Hartnett, Gerard, & Conboy, Kieran. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems, 15*(2), 200-213. doi: 10.1057/palgrave.ejis.3000605

Floyd, Christiane. (1992). *Software development as reality construction*: Springer.

Galliers, R.D. (1991). *Choosing Information Systems Research Approaches in Information Systems Research*: Alfred Waller.

Giachetti, Ronald E., Martinez, Luis D., Sáenz, Oscar A., & Chen, Chin-Sheng. (2003). Analysis of the structural measures of flexibility and agility using a measurement theoretical framework. *International Journal of Production Economics, 86*(1), 47-62. doi: http://dx.doi.org/10.1016/S0925-5273(03)00004-5

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. *Information & Software Technology, 46*(4), 243. doi: 10.1016/j.infsof.2003.07.002

Highsmith, J. (2010). *Agile Project Management* (2nd ed.). Boston, MA: Pearson Education, Inc.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer, 34*(9), 120-127. doi: 10.1109/2.947100

Iivari, Juhani, & Iivari, Netta. (2011). The relationship between organizational culture and the deployment of agile methods. *Information & Software Technology, 53*(5), 509-520. doi: 10.1016/j.infsof.2010.10.008

Janson, Marius A. Smith L. Douglas. (1985). Prototyping for Systems Development: A Critical Appraisal. *MIS Quarterly, 9*(4), 305-316.

Karlstrom, Daniel, & Runeson, Per. (2005). Combining Agile Methods with Stage-Gate Project Management. *IEEE Software, 22*(3), 43-49.

Kettunen, Petri, & Laanti, Maarit. (2005). How to steer an embedded software project: tactics for selecting the software process model. *Information & Software Technology, 47*(9), 587-608. doi: 10.1016/j.infsof.2004.11.001

Klein, Heinz K., & Myers, Michael D. (1999). A SET OF PRINCIPLES FOR CONDUCTING AND EVALUATING INTERPRETIVE FIELD STUDIES IN INFORMATION SYSTEMS. *MIS Quarterly, 23*(1), 67-93.

Laanti, Maarit, Salo, Outi, & Abrahamsson, Pekka. (2011). Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. *Information & Software Technology, 53*(3), 276-290. doi: 10.1016/j.infsof.2010.11.010

Lyytinen, Kalle, & Rose, Gregory M. (2005). How Agile is Agile Enough? Toward a Theory of Agility in Software Development *Business Agility and Information Technology Diffusion* (pp. 203-225): Springer.

Maruping, Likoebe M., Venkatesh, Viswanath, & Agarwal, Ritu. (2009). A Control Theory Perspective on Agile Methodology Use and Changing User Requirements. *Information Systems Research, 20*(3), 377-399.

Mathiassen, Lars, & Pries-Heje, Jan. (2006). Business agility and diffusion of information technology. *European Journal of Information Systems, 15*(2), 116-119. doi: 10.1057/palgrave.ejis.3000610

McAvoy, John, Nagle, Tadhg, & Sammon, David. (2013). Using mindfulness to examine ISD agility. *Information Systems Journal, 23*(2), 155-172. doi: 10.1111/j.1365-2575.2012.00405.x

McHugh, Orla, Conboy, Kieran, & Lang, Michael. (2012). Agile Practices: The Impact on Trust in Software Project Teams. *IEEE Software, 29*, 71-76. doi: 10.1109/MS.2011.118

Mohan, Kannan, Ramesh, Balasubramaniam, & Sugumaran, Vijayan. (2010). Integrating Software Product Line Engineering and Agile Development. *IEEE Software, 27*(3), 48-55.

Muthitacharoen, Achita, & Saeed, Khawaja A. (2009). Examining User Involvement in Continuous Software Development (A case of error reporting system). *Communications of the ACM, 52*(9), 113-117.

Orr, Ken. (2004). Agile requirements: Opportunity or oxymoron? *IEEE Software*, *21*(3), 71-73.

Persson, John Stouby, Mathiassen, Lars, & Aaen, Ivan. (2012). Agile distributed software development: enacting control through media and context. *Information Systems Journal*, *22*(6), 411-433. doi: 10.1111/j.1365-2575.2011.00390.x

Port, Daniel, & Bui, Tung. (2009). Simulating mixed agile and plan-based requirements prioritization strategies: proof-of-concept and practical implications. *Eur J Inf Syst*, *18*(4), 317-331.

Pozzebon, Marlei, Petrini, Maira, de Mello, Rodrigo Bandeira, & Garreau, Lionel. (2011). Unpacking researchers' creativity and imagination in grounded theorizing: An exemplar from IS research. Information and Organization. (21(4)), 177-193. doi: http://dx.doi.org/10.1016/j.infoandorg.2011.09.001

Qumer, A., & Henderson-Sellers, B. (2008). A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software*, *81*(11), 1899-1919. doi: http://dx.doi.org/10.1016/j.jss.2007.12.806

Qureshi, M. Rizwan Jameel. (2012). Agile software development methodology for medium and large projects. *IET Software*, *6*(4), 358-363. doi: 10.1049/iet-sen.2011.0110

Ramesh, Balasubramaniam, Cao, L. A. N., Mohan, Kannan, & Peng, X. U. (2006). CAN DISTRIBUTED SOFTWARE DEVELOPMENT BE AGILE? *Communications of the ACM*, *49*(10), 41-46.

Ramesh, Balasubramaniam, Lan, Cao, & Baskerville, Richard. (2010). Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, *20*(5), 449-480. doi: 10.1111/j.1365-2575.2007.00259.x

Ramesh, Balasubramaniam, Mohan, Kannan, & Lan, Cao. (2012). Ambidexterity in Agile Distributed Development: An Empirical Investigation. *Information Systems Research*, *23*, 323-339. doi: 10.1287/isre.1110.0351

Rogers, Everett M. (2003). *Diffusion of Innovations* (5th ed.). New York: Free Press.

Ronkainen, Jussi, & Abrahamsson, Pekka. (2003). Software Development under Stringent Hardware Constraints: Do Agile Methods Have a Chance? In Michele Marchesi & Giancarlo Succi (Eds.), *Extreme Programming and Agile Processes in Software Engineering* (Vol. 2675, pp. 73-79): Springer Berlin Heidelberg.

Royce, Winston W. (1970, August). *Managing the development of large software systems.* Paper presented at the proceedings of IEEE WESCON.

Salo, O., & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET Software*, *2*(1), 58-64. doi: 10.1049/iet-sen:20070038

Schatz, Bob, & Abdelshafi, Ibrahim. (2005). Primavera Gets Agile: A Successful Transition to Agile Development. *IEEE Software, 22*(3), 36-42.

Sheffield, Jim, & Lemétayer, Julien. (2013). Factors associated with the software development agility of successful projects. *International Journal of Project Management, 31*(3), 459-472. doi: 10.1016/j.ijproman.2012.09.011

Sjøberg, Dag I. K., Johnsen, Anders, & Solberg, Jørgen. (2012). Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software, 29*(5), 47-53. doi: 10.1109/MS.2012.110

Smith, Michael, Miller, James, Huang, Lily, & Tran, Albert. (2009). A More Agile Approach to Embedded System Development. *IEEE Software, 26*(3), 50-57.

Ståhl, Daniel, & Bosch, Jan. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems & Software, 87*, 48-59. doi: 10.1016/j.jss.2013.08.032

Stankovic, John A. (1996). Strategic directions in real-time and embedded systems. *ACM Comput. Surv., 28*(4), 751-763. doi: 10.1145/242223.242291

Strauss, Anselm; Corbin, Juliet. (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park, CA: Sage.

Sue, Kong, Kendall, Julie E., & Kendall, Kenneth E. (2012). PROJECT CONTEXTS AND USE OF AGILE SOFTWARE DEVELOPMENT METHODOLOGY IN PRACTICE: A CASE STUDY. *Journal of the Academy of Business & Economics, 12*(2), 1-15.

Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system. *International Journal of Production Research, 15*(6), 553-564. doi: 10.1080/00207547708943149

Svahnberg, Mikael, Gorschek, Tony, Feldt, Robert, Torkar, Richard, Saleem, Saad Bin, & Shafique, Muhammad Usman. (2010). A systematic review on strategic release planning models. *Information and Software Technology, 52*(3), 237-248. doi: http://dx.doi.org/10.1016/j.infsof.2009.11.006

Tedre, MattiSutinen Erkki. (2008). Three traditions of computing: what educators should know. *Computer Science Education, 18*(3), 153-170. doi: 10.1080/08993400802332332

Van de Ven, A. . (2007). *Engaged Scholarship: A Guide for Organizational and Social Research*. New York, NY: Oxford University Press.

Vidgen, Richard (2009). Coevolving Systems and the Organization of Agile Software Development. *Information Systems Research, 20*(3), 355-376.

Vinekar, Vishnu, Slinkman, Craig W. , & Nerur, Sridhar. (2006). CAN AGILE AND TRADITIONAL SYSTEMS DEVELOPMENT APPROACHES COEXIST? AN AMBIDEXTROUS VIEW. *Information Systems Management, 23*(3), 31-42.

Vlaanderen, Kevin, Jansen, Slinger, Brinkkemper, Sjaak, & Jaspers, Erik. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and Software Technology, 53*(1), 58-70. doi: http://dx.doi.org/10.1016/j.infsof.2010.08.004

Wang, Xiaofeng, Conboy, Kieran, & Cawley, Oisin. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems & Software, 85*(6), 1287-1299. doi: 10.1016/j.jss.2012.01.061

Wang, Xiaofeng, Conboy, Kieran, & Pikkarainen, Minna. (2012). Assimilation of agile practices in use. *Information Systems Journal, 22*(6), 435-455. doi: 10.1111/j.1365-2575.2011.00393.x

Yauch, Charlene. (2011). Measuring agility as a performance outcome. *Journal of Manufacturing Technology Management, 22*(3), 384-404.

Yin, Robert K. (2009). *Case Study Research Design and Methods*. Thousand Oaks, CA: Sage.