An Effectual Approach for the Development of Novel Applications on Digital Platforms

Onkar Shamrao Malgonde
University of South Florida, omalgonde@usf.edu

Follow this and additional works at: https://digitalcommons.usf.edu/etd

Part of the Databases and Information Systems Commons

Scholar Commons Citation

This Dissertation is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact digitalcommons@usf.edu.
An Effectual Approach for the Development of Novel Applications on Digital Platforms

by

Onkar Shamrao Malgonde

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Information Systems & Decision Sciences
Muma College of Business
University of South Florida

Major Professor: Alan R. Hevner, Ph.D.
Balaji Padmanabhan, Ph.D.
Richard Will, Ph.D.
Diana Hechavarria, Ph.D.

Date of Approval:
June 14, 2018

Keywords: effectual software development, digital innovation, platforms, effectuation

Copyright © 2018, Onkar S. Malgonde
DEDICATION

This dissertation is dedicated to Roshni V. Dulal with love.
ACKNOWLEDGMENTS

In completing this dissertation, I have benefited greatly over these years from discussions with Dr. Alan R. Hevner. I would like to acknowledge his support and guidance through my Ph.D. journey and thank him for countless reviews, suggestions, and motivation across these years. Dr. Hevner, a true mentor, has been instrumental in my development as a researcher and teacher.

I would also like to acknowledge the support and guidance of Dr. Rosann Webb Collins in the conceptualization, data gathering and analysis, and presentation of this research. I have greatly benefited and enjoyed the discussions with Dr. Rosann. I would like to thank her for her insights on matters beyond academic work.

I would like to acknowledge the support of my dissertation committee and other faculty members in the Information Systems and Decision Science department at the Muma College of Business for providing an environment conducive for learning and research. I would also like to acknowledge the support of Dr. Matthew Mullarkey for introducing the case study sites and the financial support of L. Rene Gaiennie endowment.

I would like to recognize the patience and support of my family. My parents, Shamrao and Sanjivani Malgonde, for their encouragement and sacrifices. My parents-in-law, Vinod and Vasudha Dulal, for their patience, motivation, and support. My sister, Monika Patel, brother-in-law, Manoj Patel, for their support and encouragement over these years. I also acknowledge the efforts of Rohit Dulal in keeping my spirits high through these years. Finally, I would like to
acknowledge the support of my wife, Roshni V. Dulal. This work has benefited from discussions and feedback from Roshni.
# TABLE OF CONTENTS

LIST OF TABLES .......................................................................................................................... v

LIST OF FIGURES ....................................................................................................................... vi

ABSTRACT .................................................................................................................................. vii

Chapter 1. INTRODUCTION .................................................................................................. 1
  1.1 Digital Innovations on Software Platforms ................................................................. 2
  1.2 Theoretical Basis ......................................................................................................... 6
  1.3 Research Approach ................................................................................................. 7
  1.4 Research Question ................................................................................................... 8
  1.5 Findings and Contributions ................................................................................... 9
  1.6 Summary ................................................................................................................... 11

Chapter 2. LITERATURE REVIEW AND THEORY .......................................................... 12
  2.1 Software Platforms ................................................................................................... 12
      2.1.1 Locus of Innovation ..................................................................................... 12
      2.1.2 Software Development Challenges ............................................................ 15
  2.2 Software Development Approaches ....................................................................... 16
      2.2.1 Portfolio of Controls in Software Development .......................................... 18
      2.2.2 Risk Analysis in Information Systems projects ......................................... 20
  2.3 Information Systems and Entrepreneurship .......................................................... 22
  2.4 Theory of Effectuation .......................................................................................... 23
  2.5 Prediction vs Control in Software Development ................................................... 25
  2.6 Effctual Software Development ............................................................................. 28
  2.7 Summary .................................................................................................................. 31

Chapter 3. AN EFFECTUAL APPROACH TO SOFTWARE DEVELOPMENT ........ 32
LIST OF TABLES

Table 1. Constructs, First Cycle Codes, and Operational Definitions ........................................ 39
Table 2. Constructs and their Frequency in the Apache Cordova Data ........................................ 42
Table 3. Summary of Case Study Locations ............................................................................. 66
Table 4. Summary of the Project – AT Case Study ............................................................... 67
Table 5. Summary of the Project – TB Case Study ............................................................... 68
Table 6. Summary of Interviews Transcribed ........................................................................ 71
Table 7. Summary of Interviews Coded .................................................................................. 73
Table 8. Frequency of Construct Coding ................................................................................ 74
Table 9. Key Points in AT Interviews ..................................................................................... 77
Table 10. Key Points in TB Interviews ..................................................................................... 79
Table 11. Cross-Case Analysis ............................................................................................... 80
Table 12. Comparison of Applications’ Nature ...................................................................... 86
Table 13. Software Development Approaches ..................................................................... 147
Table 14. AT Case Findings .................................................................................................. 164
Table 15. TB Case Findings .................................................................................................. 172
LIST OF FIGURES

Figure 1. Digital Platform Ecosystem ............................................................................................. 2
Figure 2. Software Development Approaches (adapted from Harris et al. (2009a)) .................... 17
Figure 3. Theory of Effectuation (Sarasvathy 2001) .................................................................... 23
Figure 4. Framework of Prediction and Control (Wiltbank et al. 2006) ....................................... 26
Figure 5. Software development approaches and the framework of prediction and control ...... 27
Figure 6. Components and Relationships of the Effectual Development Process ...................... 33
Figure 7. Apache Cordova Architecture ....................................................................................... 37
Figure 8. Revised Model of Effectual Software Development Process ........................................ 48
Figure 9. Research Design ............................................................................................................ 61
Figure 10. AT’s Development Process ......................................................................................... 75
Figure 11. TB’s Development Process .......................................................................................... 78
Figure 12. Prediction and Control of AT’s team members .......................................................... 123
Figure 13. Prediction and Control of TB’s team members ......................................................... 125
Figure 14. Two phases of Effectual Software Development Process ........................................ 126
Figure 15. Flow of Planning Activities ....................................................................................... 127
Figure 16. Flow of Execution and Monitoring Activities ........................................................... 128
ABSTRACT

The development of novel software applications on digital platforms differs from traditional software development and provides unique challenges to the software development manager and team. Application producers must achieve application-platform match, application-market match, value propositions exceeding platform’s core value propositions, and novelty. These desired properties support a new vision of the software development team as entrepreneurs with a goal of developing novel applications on digital platforms. Digital platforms are characterized by an uncertain, risky, and resource-constrained environment, where existing approaches—plan-driven, ad-hoc, and controlled-flexible—have limited applicability. Building on the theoretical basis of the theory of effectuation from the entrepreneurship domain, this dissertation proposes an effectual approach to software development. Preliminary studies are conducted to provide prima facia evidence of effectual thinking in software development teams. Also, pilot interviews at local organizations are conducted to augment the approach. Finally, two case studies are conducted to validate the approach. We find conclusive evidence for the efficacy of effectual software development to develop novel applications on digital platforms. We also find that novel ideas are identified, honed, and incorporated, in the application, using effectual thinking. This study contributes to information systems literature by proposing and validating an effectual approach to software development. This study contributes to entrepreneurship literature by illustrating the role of planning and visionary approaches in effectuation settings. This study also contributes to practitioners by highlighting the theoretical underpinnings of existing approaches.
and the effectual approach which allows software development teams to incorporate effectual thinking and develop novel software applications. Finally, we conclude with a discussion on the theoretical contributions of this study, limitations, and future research avenues.
CHAPTER 1. INTRODUCTION

Software development is characterized by four dimensions: technology, people, process, and product. A team of software development professionals (including developers, testers, architects, designers, project managers) identify suitable technology (tools, programming language, hardware, software) to develop a product by following a process. Prior work in information systems has considered portfolio of controls (Harris et al. 2009b) and different approaches to software development (Harris et al. 2009a) and their applicability to fast-moving environments. Competing products may undertake different decisions for each dimension which offer a source of differentiation for the team.

Recent proliferation of digital platforms significantly alters the dynamics of software development dimensions. Digital platforms provide abstract capabilities to software development teams via standardized connection interfaces. Sources of uncertainty are expanded to include users’ preferences, competing platforms, and competing applications on the platform. The digital platform and its environment expand the challenges considered in prior work and highlight resource-constrained, risk, uncertainty, novelty and value proposition requirements of the application. To understand these challenges, we discuss the digital platform and components of its ecosystem.
1.1 Digital Innovations on Software Platforms

Digital innovations are new combinations of digital and physical components characterized by re-programmability, homogenization of data, and use of digital technology (Yoo et al. 2010). The digital platform, as shown in Figure 1, is a pervasive digital technology that is rapidly transforming the ways in which products and services are produced and consumed in our market economy (Parker et al. 2016).

The platform ecosystem consists of the platform, applications that are available via the platform or connect to the platform via the interfaces offered by the platform, in a contextual environment of regulations and competitors (Tiwana 2013; Tiwana et al. 2010). Platforms enable value-creating interactions among organizations with disparate resources and specializations (Parker et al. 2016). This transfers the locus of innovation, which traditionally has been within the organization, to a diverse set of external organizations that develop applications available via the platform. The Platform owner is the organization or group of organizations that determine the architecture, governance, and curation mechanisms for the platform. Producers are the organizations that develop applications (extensions to the core functionality offered by the platform) that are available via the platform. Consumers are the organizations that use

Figure 1. Digital Platform Ecosystem

Digital innovations are new combinations of digital and physical components characterized by re-programmability, homogenization of data, and use of digital technology (Yoo et al. 2010). The digital platform, as shown in Figure 1, is a pervasive digital technology that is rapidly transforming the ways in which products and services are produced and consumed in our market economy (Parker et al. 2016).

The platform ecosystem consists of the platform, applications that are available via the platform or connect to the platform via the interfaces offered by the platform, in a contextual environment of regulations and competitors (Tiwana 2013; Tiwana et al. 2010). Platforms enable value-creating interactions among organizations with disparate resources and specializations (Parker et al. 2016). This transfers the locus of innovation, which traditionally has been within the organization, to a diverse set of external organizations that develop applications available via the platform. The Platform owner is the organization or group of organizations that determine the architecture, governance, and curation mechanisms for the platform. Producers are the organizations that develop applications (extensions to the core functionality offered by the platform) that are available via the platform. Consumers are the organizations that use
applications offered via the platform. Further, consumers can mix-and-match applications available via the platform to satisfy their need. Examples of software platforms include Apple’s iTunes, Google’s Play, Salesforce’s appexchange, SAP’s HANA, Valve’s Steam, and Instructure’s Canvas, among others.

Development of novel software applications on a digital platform differs from traditional software development. The goals of this dissertation are to understand these key differences and to propose innovative application development processes for digital platforms. The following platform characteristics support a new and challenging application development environment:

- A platform offers a compelling set of core value propositions to its consumers (Parker et al. 2016). Applications on the platform play off the core values and add novel extensions to the platform’s capabilities.
- Over time, these core values evolve based on consumer demands and goals and, as a result, platform applications are added, updated, and dropped.
- As the number of similar applications on a software platform increases, investment incentives for individual producers are crowded out (Boudreau 2012). Similarity of applications available via a platform limits the platform’s value proposition and incentivizes the platform to assimilate those features into the core value proposition of the platform. Consequently, applications whose value proposition is assimilated into the core offering of the platform are discontinued.
- All applications must adhere to connection specifications and development procedures determined by the platform (Tiwana 2013). Platforms provide standard connection interfaces in the form of application programming interfaces (API’s) that are used by applications to access common features within the platform. Thus, platform owners and user groups often
require that application producers follow certain best practices such as ‘look and feel’
interactions. In many cases, the platform owners evaluate and approve new applications
(curation mechanisms) before they are offered to consumers via the platform.

- Application developers may request changes in platform interfaces and protocols based on
environmental changes or new customer demands.
- Platforms exhibit different levels of maturity over time. Changes to platform architecture and
governance mechanisms requires application producers to adapt their applications and
routines to comply with updated platform regulations.

To manage these unique challenges and provide value-added applications, producers must
achieve (a) application-platform match, (b) application-market match, (c) value propositions
exceeding platform’s core value propositions, and (d) novelty. An application is valuable to
platform consumers if it provides features and extensions that can enable consumers to perform
activities that the platform does not provide. Further, an application is novel if it provides
features and extensions that the platform and other applications do not provide\(^1\). These desired
properties support a new vision of the software development team as entrepreneurs with a goal of
developing novel applications on digital platforms.

Prior research in software application development largely focuses on the desired properties
of application-market match and project performance (Weiner et al. 2016). However, the success

---

\(^1\) These follow the accepted definitions of value and novelty in software development context as used by Austin and Devin
(2009) who build on extant literature related to new product development in information systems, business, and psychology.
criteria for software applications on digital platforms\(^2\) significantly exceeds these traditional properties since the environment provided by digital platforms is richer and more complex (McKelvey et al. 2015). The desired properties of application-platform match, application-market match, value propositions exceeding platform’s core value propositions, and novelty for an application on digital platform support a new vision of the software development team as entreprenuers. Current thinking on agile software development can be extended via the development of an effectual approach to software development that is appropriate in environments characterized by uncertainty, risk, resource-constraints, and nascent markets. This dissertation proposes and validates an effectual approach to software development that is grounded in the theory of effectuation from the entrepreneurship domain.

The effectual approach incorporates effectual thinking by identifying possible action possibilities and intermediate effects based on its means and aspirations. Intermediate effects allow the software development team to evaluate its current understanding and identify new resources and attenuate its aspirations. Effectual thinking involves following the fast and tight design cycles to identify intermediate effects and iteratively realize the final artifact. The effectual approach differs from causation-based approach prevalent in current software development projects. Providing a succinct explanation of the difference between causation and effectuation, Sarasvathy (2001) notes “Imagine a chef assigned the task of cooking dinner. There are two ways the task can be organized. In the first, the host or client picks out a menu in

\(^2\) Extant work has considered software development for platforms (technological platform that define specific requirements for the development team). However, the platform considered in prior work does not include the uncertainty and risk characterized by third-party ownership of the platform, competing firms on the platform, and novelty of the application in focus.
advance. All the chef needs to do is list the ingredients needed, shop for them, and then actually cook the meal. This is a process of causation. It begins with a given menu and focuses on selecting between effective ways to prepare the meal. In the second case, the host asks the chef to look through the cupboards in the kitchen for possible ingredients and utensils and then cook a meal. Here, the chef has to imagine possible menus based on the given ingredients and utensils, select the menu, and then prepare the meal. This is a process of effectuation. It begins with given ingredients and utensils and focuses on preparing one of many possible desirable meals with them.” (p. 245). This research extends the effectuation thinking in software development projects to develop novel applications on digital platforms.

1.2 Theoretical Basis

The theoretical basis for this dissertation comes from three research streams: software platforms, software development approaches, and entrepreneurship. Literature from the software platform research stream is used to identify key challenges which are different from the challenges studied in prior work in software development for developing novel software applications on digital innovations like software platforms. These challenges stem from the architecture and governance mechanisms of the platform ecosystem and require the software development approach to proactively shape and evolve the product to be relevant in the future.

To address these challenges, we explore the underlying logic of existing software development approaches using the framework of control and prediction (Wiltbank et al. 2006). The framework shows that existing software development approaches focus on positioning the software product in an exogenous environment so that it is relevant (and profitable) in the future. Positioning is achieved by predicting the environment or adapting to changes in the environment. Given the challenges offered by platform ecosystem and the positioning strategy showcased by
existing software development approaches, existing software approaches fall short of satisfying the desired properties of application-platform match, application-market match, value propositions exceeding platform’s core value propositions, and novelty.

These limitations support a new vision of the software development team as entrepreneurs with a goal of developing novel applications on digital platforms. We turn to the theory of effectuation, from entrepreneurship domain, to understand the processes adopted by entrepreneurs and propose an effectual approach to software development. The effectual approach is further informed by control modes and risk analysis in software development, with theoretical groundings in control theory (Ouchi 1979) and decision making (March and Shapira 1987), respectively. Chapter 2 describes the extant literature on the three research streams.

1.3 Research Approach
Based on the prior work in software development and the theory of effectuation, a model of effectual approach is developed to identify key constructs and relationships between the constructs. The model identifies platform characteristics, software development team’s means, aspirations, actions, and feedback loops from intermediate products. To garner support for presence of effectual thinking in novel software development projects, a qualitative study using secondary data from an open source software development project is analyzed. The open source project (Apache Cordova) supports a novel application development and allows software development teams to develop mobile applications that can be deployed across different platforms. We find prima facie evidence of effectual thinking in the open source software development projects. To further understand and operationalize constructs and feedback processes identified in the model, we conduct pilot interviews with two managers of software development projects which are developing applications on digital platforms. Analysis of these
pilot interviews and secondary data help us to define first cycle codes for constructs and provide operational definitions for the first cycle codes. The analysis also helps us to identify new constructs and revise relationships in the model.

To further validate the model, we conduct qualitative case studies at two companies that are developing novel software application on digital platforms. The focus of these interviews was to distil the software development approach followed by the teams, role of platform, and heuristics used to identify actions. The unit of analysis was the software development project. Analysis of interview data include first cycle-coding using two independent coders and axial coding to identify broader themes in the data. We find support for the revised model in the primary qualitative data. Finally, we map the individual roles of team members to the broader theme of control and/or prediction.

1.4 Research Question

This dissertation focuses on the development of novel software applications on digital platforms. The broader question considered in this research is: “What software development methods best support software project teams to design, build, evaluate, and deploy novel applications on digital platforms?” The framework of control and prediction helps us to consider different software development approaches and their suitability to application development on digital platforms. Following this broader research question, this dissertation also considers the question: “How do software development teams incorporate effectual thinking in the development of novel applications on digital platforms?” The research model provides theoretical explanation of the effectual process followed by software development teams.
1.5 Findings and Contributions

This dissertation provides three findings to the literature in software development and entrepreneurship. First, this dissertation provides evidence for effectual thinking in software application development projects on digital platforms. Effectual thinking is appropriate in the resource-constrained and risky environment characterized by digital platforms. Second, software development team members assume different roles and related responsibilities. With their role, team members have different tasks and views of the development process. For example, developers are focused on their module whereas technical architect will consider technical challenges across different modules and interfaces of the application. Different tasks require different configuration of portfolio of controls. We find that different roles in the software development team will emphasize control and/or prediction differently. Specifically, we find that team members in leadership roles emphasize prediction over control in certain areas such as choice of platform, technology, novelty of the application. Often, such prediction-based approach is driven by organizational and market aspects. Similarly, team members emphasize control over prediction in areas where uncertainty is high, and team’s knowledge is limited. Finally, we also find that the emphasis on control over prediction may change with different phases of the application development. Initial stages are usually characterized by prediction whereas subsequent phases are characterized by control and limited prediction. These findings have implications for entrepreneurship literature which consider control or prediction to be the sole choice over the product development lifecycle.

This dissertation aims to provide three theoretical contributions to the literature in software development and entrepreneurship. First, building on the challenges identified for the development of novel applications on software platforms, this dissertation advances a new vision
of software development where the software development project is envisioned as an entrepreneurial endeavor and project manager and development team as entrepreneurs. Second, an effectual approach to development of novel applications on software platforms has been described. Grounded in the theory of effectuation, the approach introduces context specific constructs (platform, nature of application, actions, intermediate effects) and theorizes and adapts existing effectuation constructs to the software development context. Third, according to Whetten (1989), one of the critical criteria to judge theoretical contributions is its ability to introduce theoretical insights that influences existing understanding of the phenomenon. Effectual approach to software development introduces new constructs and feedback processes in software development research – aspirations, focus on existing resources, decision heuristics, expanding and converging cycles. These effectual processes provide improved explanations for novel application development on software platforms where existing approaches have failed.

This dissertation also contributes to practice. First, we draw attention to the development approaches for novel applications on software platforms which has received limited attention in the information systems literature. Attention to development approaches on software platforms is particularly important and timely, given the proliferation of platforms (Parker et al. 2016). Second, application producers have a direct interest in development approaches that specifically address the unique challenges offered by platform ecosystems. These interests extend beyond development of novel applications and include development of inimitable applications and maintenance of existing applications. Third, platform owners also benefit from the introduction of novel applications on software platforms. As the locus of innovation shift from within the organization to a heterogeneous base of application producers, introduction of novel application
allows the platform to serve diverse consumer segments and introduce new demand within the user group.

1.6 Summary

Digital platforms have transformed the way in which products and services were offered. These platforms also identify new challenges to software application development teams. Prior focus on product-market match and project performance as key success criteria for software applications has been extended to include product-platform match, exceeding platform’s value proposition, and novelty of the application. Prior work in information systems has not considered the issues pertaining to the software development team on digital platforms. Existing software development approaches have limited applicability to the resource-constrained and risky environment characterized in digital platforms.

This dissertation identifies the limited applicability of existing software development approaches to novel software application development on digital platform and draws on entrepreneurship literature to identify an effectual approach to software development. The effectual approach extends existing approaches by highlighting the need for effectual thinking and identifies avenues for software development teams to incorporate effectual thinking in their processes.
CHAPTER 2. LITERATURE REVIEW AND THEORY

This chapter describes the literature review and theoretical groundings for the dissertation. First, we briefly discuss the prior work on digital platforms and the characteristics which develop the challenges for software development teams. Second, we discuss the literature on software development approaches and their applicability towards development of novel software applications. In the next subsection, we discuss literature on software platforms in IS and organization science to identify a new vision of software development. Third, in proposing a novel approach to software development on platforms, we discuss the theory of effectuation from entrepreneurship domain which provides the theoretical basis for this study. Next, this chapter describes the underlying logic of existing software development approaches and discuss their limitations to applicability towards software development on platforms. Finally, the literature on portfolio of controls and risk analysis in software development is discussed.

2.1 Software Platforms

2.1.1 Locus of Innovation

Traditionally, organizations developed innovative product lines through a linear value chain (Parker et al. 2016). Products were designed, developed, and marketed by a single firm. However, with the pervasive digital innovations and technology, the locus of organizational innovation has shifted to digital software platforms. Software platforms are digital innovations that allow combination of digital and physical components to produce novel products (Yoo et al. 2010). For example, e-book readers (Kindle, iPad) – physical component - allows storage,
editing, sharing, and access to thousands of e-books – digital component. As a consequence, organizations can create innovations that are characterized by convergence\textsuperscript{3} and generativity\textsuperscript{4} (Yoo et al. 2012). There are three perspectives associated with platforms (Eaton et al. 2015; Yoo et al. 2012) : generative, infrastructure, and platform view. The first perspective is known as the generative view (Eaton et al. 2015; Yoo et al. 2010). This perspective focuses on the transition of organizational innovation from dominated by single organization (Faraj et al. 2011) to participation of heterogeneous actors. Heterogeneous actors combine existing resources and create innovations that are further used by other actors to create innovations of their own. Thus, innovations build on each other, provide alternative innovations, and evolve in direction and magnitude, unforeseen by platform creators. Examples of this view include app store, on-line communities, and social media.

The second perspective is the infrastructure view. According to this view, a central firm creates and controls the infrastructure (platform) that acts as an anchor for third-party producers to build applications. The creator exerts considerable control on the evolution of the platform. Typically, this view exists in industries where infrastructure creation and maintenance is expensive and high barriers to entry exist. For instance, inventory management systems provided digital trace of physical processes. Due to digital innovations, enterprise resource planning (ERP) systems significantly altered organizational control of inventory management (Tilson et al.

\textsuperscript{3} Convergence is the ability of the innovation to attract disparate services, functionalities, and industries.

\textsuperscript{4} Generativity is “a technology’s overall capacity to produce unprompted change driven by large, varied, and uncoordinated audiences”\textsuperscript{(pg. 1980)} (Zittrain 2006)
Other tools and subsystems can then be designed by third-party organizations to interact with the established ERP systems and add value to the core platform.

Finally, the third perspective is the platform view. This perspective views platforms as ecosystems where core resources are created and managed by firms, and producers create applications that are consumed by consumers. The platform allows value creation and exchange between platform owner, producers, consumers, and environment. Each actor exerts considerable influence on the platform. Further, the environment plays a crucial role in enhancing, constraining, and evolving the platform. For instance, software-based platforms, as an ecosystem, provide core functionality, governance, and interfaces to modules (subsystems) that connect and utilize the core functionality of the platform and provide value to users (Tiwana et al. 2010). Consider the ‘chrome web store’ platform for Google’s Chrome browser. The platform provides thousands of apps, games, extensions, and themes (modules) to consumers that range from free to paid and are developed by third-party producers. They use standard programming languages and application programming interface (APIs) set forth by the platform. Further, platforms have extensive curation devices in place that allow platform owners (Google Chrome) to screen potential products (apps, games, or themes) before they are available to platform consumers.

With these views, the focus on organizational innovations shifts from within the organization to the interfaces and outside the organizational boundaries. Yoo et al. (2012) discuss multiple implications of this view for organizations in general. First, platform owners must balance generativity and control on the platform. These opposite yet required characteristics are important to achieve generative growth by involving third-party producers while controlling the direction of platform’s evolution. Second, in comparison to traditional
restrictions of limited data and process access to entities outside firm boundaries, platform organizations are increasingly sharing data and processes across its boundaries. This allows innovations and associated activities to be increasingly horizontal. That is, innovative products and their components can be used across different platforms, customer base, and other products. Third, with a shift of focus from within the organization to its boundaries and beyond, innovations require heterogeneous knowledge sources that are outside organizational boundaries and are integrated temporarily. Fourth, platforms give rise to different forms of risks when major activities are outside firm boundaries. Finally, platforms allow combinatorial innovations where modular components can be mashed up to create incremental innovations. Such innovations require new forms of innovation processes and creativity (constrained serendipity). A recurring theme across these views is the shift of locus of innovation towards a wider community of application producers.

### 2.1.2 Software Development Challenges

Prior work in software development has considered product-market match as the critical success criteria for software application development. To satisfy this success criteria, software development teams could draw upon different technologies and tools that the team would deem appropriate for the application to be developed. In the case of application development on platforms, software development teams have a limited set of technologies and tools to choose from for the application. The limited set of technologies and tools are compatible with the platform. Choice of non-compatible technologies and tools will require development and maintenance of adaptors.

To address the limited set of technologies and tools available to application developers, platforms offer standard APIs that can be easily used by application developers. These APIs
encapsulate common functionality required in applications and allow application developers to focus on niche features in their application. The set of standard APIs is available to all the application development teams. Thus, competing applications cannot differentiate themselves based on the common set of APIs offered by the platform.

In a platform environment, applications compete on the application’s content, features, quality, and performance. These competing factors require application development teams to identify application’s components and features that will set it apart from other competing applications in the market and platform. Thus, the success criteria for software applications on platforms go beyond application-market match (Harris et al. 2009a), to include application-platform match, value propositions exceeding platform’s core value propositions, and novelty. In what follows, we will review prior literature on existing software development approaches and the trade-offs that they address. Following which we discuss the underlying logic of existing software development approaches to determine their applicability to the development of novel applications of software platforms.

2.2 Software Development Approaches

The central tension in software development is the balance between control and flexibility. Although there exist a number of software development methods favoring planning over flexibility and vice versa, these methods can be broadly classified into three paradigms - plan-driven, ad-hoc, or controlled-flexible (Harris et al. 2009a). Figure 2 (Harris et al. 2009a) illustrates the trade-off. A plan-driven approach to software development emphasizes planning to streamline the development process by predicting future state of the environment and positioning the product for relevance. Initially, the development effort focuses on identifying and analyzing user requirements. User requirements are analyzed to determine their feasibility and profitability
of the software product upon its completion. If deemed feasible, architecture and design for the software product is developed, followed by actual product development, testing, and release. Subsequent releases incorporate user feedback and fix newly identified defects. Plan-driven approaches lend themselves to the development of novel software products in stable environments where detailed specifications can be developed a priori, and maintain their relevance after the product is developed. Eisenhardt and Tabrizi (1995) found empirical evidence of linear development processes in moderately dynamic environments in the computer industry. MacCormack and Verganti (2003) find similar evidence in internet software development projects. Further, plan-driven approaches have been identified as the dominant approach in new product development literature (Nambisan 2003) where opportunity for economic profit can be identified and realized (Shane and Venkataraman 2000).

An organic ad-hoc approach to software development emphasizes persistent adaptation to changing environment. Minimal specifications are developed, if at all, as team members self-monitor their efforts to develop the software product. However, these approaches advance just-in-time solutions that are relevant for the given situation and do not scale to produce relevant outcomes (Harris et al. 2009a).
Controlled-flexible approaches balance planning and flexibility under increasing uncertainty in market and technology. The initial project landscape provides partial specification of the product to be developed. Adapting to environmental uncertainty, feedback mechanisms allow the specification to be modified so that the product can match changing market needs. Limiting the solution space, scope boundaries allow exploration of solution by the project team while constraining their space to align with organizational goals. Thus, iterative development, scope boundaries, and feedback mechanisms allow the software product to be relevant upon completion (Harris et al. 2009a). Though controlled-flexible approach develops a relevant product in uncertain environment, none of the prior literature has explored its applicability to the development of novel software products.

Control modes provide the underlying mechanisms that aligns the goals of all stakeholders, across all software development approaches. Project managers identify appropriate control modes to develop a portfolio of controls that is suitable for their project. Through the composition of portfolio of controls, project managers can alter the development approach suitable for the project at hand. Following subsection describes the control modes used in software development projects.

### 2.2.1 Portfolio of Controls in Software Development

A central responsibility of any manager is to exercise control over employees and organizational activities. Control theory (Ouchi 1977; Ouchi 1979) explains different control modes available to managers, including project managers. It provides the lens that guides the development of a project-specific methodology. Control modes are categorized into two types: *formal* and *informal*. Formal modes of controls are viewed as performance rewarding strategies
by the management (Eisenhardt 1985; Kirsch 1997). In formal control mode, the management specifies a goal and rewards the team upon completion of the project goal.

Two forms of formal control are outcome control and behavior control. *Outcome control* specifies establishing a prior set of goals and determining reward levels based on the extent to which established goals have been accomplished. For example, specified software load time is a system goal. If such a load time is consistently achieved, the software team has met the outcome goal and can be rewarded based on a pre-specified contract. *Behavior control* specifies adherence to established processes that software development teams should follow in order to achieve the outcome goals. In such a control mode, management’s emphasis is on observing team’s behavior. For example, presence in daily Scrum meetings is expected from team members so that information can be shared.

In contrast to formal modes of control, an informal mode of control relies on a social strategy to achieve the goal of aligning organizational and employee goals. Two forms of informal control are clan control and self-control. *Clan control* relies on the team to foster a unique set of rules, applying to all, that help in achieving the common goal for the team. Management has limited leverage on such a control since it is loosely coupled from the organization goals and is highly influenced by interactions within the team. *Self-control* emphasizes individual autonomy to achieve goals set by the individual. In a software development team, individuals are required to be creative and govern their own individual processes to meet deadlines (Henderson and Lee 1992). In professional settings like software development informal modes of control are also influenced by developers’ education and socialization to the profession.
In order to extend Control Theory to handle situations with uncertainty, Harris et al. (2009a) propose a new mode of control: emergent outcome control (EOC). They identify two EOC mechanisms. Scope boundaries limit the feasible solution such that the development team has the flexibility to explore but is constrained within a boundary. However, the project team is unconstrained within the boundaries thereby maintaining creativity. Ongoing feedback is provided to the team, from users, or the market, to steer development so that specifications are closely met. For example, feedback can be provided to the team via meetings, documentation, user reviews, or market orientation. Such feedback allows them to adjust their development to specific needs of the market and achieve their goal. Project managers employ control mechanisms to implement control modes (Choudhury and Sabherwal 2003; Kirsch 1997). For example, delivering a working prototype every 2 weeks implements outcome control by specifying a target for every development cycle. Also, it implements behavioral control by providing a sense of urgency within the team.

Application development on software platforms introduces significant risk to application producers. In a platform ecosystem, application producers face the risk of platform obsolescence, an application’s value proposition being assimilated by core platform, application replication by rival producer, market demand, and platform interface, among others. In what follows, we describe the prior literature on risk analysis in information systems investments and projects.

2.2.2 Risk Analysis in Information Systems projects

Risk analysis in software development consist of risk assessment and risk control (Boehm 1991). In risk assessment, different risk factors are identified, analyzed, and ranked in the order of severity. In risk control, plans, resolution mechanisms, and monitoring devices are identified to address negative events. In a cross-cultural study, Schmidt et al. (2001) identify an exhaustive
list of risk factors for software development projects. These factors are further classified based on the influence the project manager has on managing the risk factors. Also, studies by Barki et al. (1993), Ropponen and Lytyinen (2000), and Keil et al. (1998), identify risk factors for software development projects.

Managing the identified risks in software development projects has been subject of research over two decades. Lytyinen et al. (1998) argue that project managers actively monitor inception of risky incidents (source of incidents may be internal or external to the project) and provide interventions. Heuristics developed through prior experience, planning, and formalized decision routines allow selection and application of appropriate intervention mechanisms. Based on the specific risk items identified, Alter and Ginzberg (1978), Davis (1982), McFarlan (1982), and Boehm (1991), offer risk reduction strategies. Building on real options view, Benaroch (2002) and Benaroch et al. (2006) develop real options that can be embedded in information technology investment that will enable the organization to control risk factor that may arise in future.

Based on the risk analysis for a software development project, decision alternatives are evaluated by the project manager. According to classical decision theory, risk associated with an alternative is the variation in its possible outcomes (March and Shapira 1987). The larger the variation in possible outcomes, the larger is the risk associated with the alternative. Thus, evaluation of decision alternatives is based on the trade-off between its expected return and associated risk. This perspective is in line with the causation logic, where decision alternatives are chosen based on their expected returns and risks. With a transformative approach, a managerial perspective on risk suggests that risk is associated with negative outcomes of the decision alternative. Alternatives are deemed risky if the loss as a result of that alternative is
high. Schmidt et al. (2001) find similar decision making among IT project managers. Further, the managerial perspective also notes that risk is *controllable* and *modifiable* through skills and information (MacCrimmon and Wehrung 1986).

### 2.3 Information Systems and Entrepreneurship

In a recent editorial commentary titled “IT and Entrepreneurism: An On-Again, Off-Again Love Affair or a Marriage?”, Del Giudice and Straub (2011) note that information technology (IT) provides “a magic ingredient” (pg. iii) that has enabled development of entrepreneurial firms. Exploring this belief, multiple studies have empirically demonstrated the key role played by emerging technology, globalization, and need to improve efficiency in providing entrepreneurial opportunities across the globe. IT development and diffusion leads to rise in productivity and an entrepreneurial culture in advanced economies (Vu 2004). Increasingly, entrepreneurial firms with an emphasis on IT related products and services accelerated economic development and influenced policies. Conversely, literature in the entrepreneurship has regularly studied entrepreneurial behavior in developing new ventures based on information technology. For instance, Fisher (2012) study six new ventures, all of which are software based enterprises, to understand the underlying logic used by the entrepreneurs. In the book *Founder’s at work*, Livingston (2007) interviews 32 founders of new ventures, all of them based on software systems.

Recently, there have been increasing call for greater collaboration between digital technologies and entrepreneurship, theorizing above and beyond the enabler role played by information technology (Nambisan 2016; Nambisan and Baron 2013). This dissertation aims to answer such a call by theorizing an effectual approach to software development for development of novel application of software platforms.
2.4 Theory of Effectuation

Sarasvathy (2001) conceptualizes effectuation\(^5\) as the opposite of causation\(^6\). Unlike causation, effectuation does not focus on finding causes that explain or achieve a given (intended) effect, but considers available actions through given means and their spectrum of possible effects. Effectuation therefore is about designing and evaluating alternatives with differing effects (and choosing one of them) instead of choosing among given alternatives which all lead to the same effect. Thus, effectuation logic constitutes a logic of controls, specifically controlling the future by actively shaping one’s environment within one’s possibilities.

In effectuation, the choice of action depends on the three, given means of 1) the actors (effectuators) themselves and their traits (“who I am”), 2) their knowledge (“what I know”), and

---

\(^5\) Effectuation is defined as a process which “takes a set of means as given and focus on selecting between possible effects that can be created with that set of means” (Sarasvathy 2001, p. 245).

\(^6\) Causation is defined as a process which “takes a particular effect as given and focus on selecting between means to create that effect” (Sarasvathy 2001, p. 245).
3) their social connections (“whom I know”). It also depends on what the effectuators can imagine to be possible effects and what they perceive the corresponding risks or potential losses to be. These risks and losses are matched with effectuators’ set of aspirations, leading to the eventual choice of action. Neither the means nor the aspirations are treated as invariant, leading to a concept that embraces flexibility and dynamism, allowing the exploitation of emerging contingencies (Sarasvathy 2001). Figure 3 illustrates the basic concepts of effectuation.

Two decision heuristics are employed when the entrepreneur pursues possible actions: acceptable risk/affordable loss and logic of control. Acceptable risk/affordable loss favors those actions which carry a degree of risk that is acceptable to the entrepreneur. It avoids actions that carry existential risk to the enterprise. This is in contrast to causation where decision making is based on expected returns of the alternative actions. Logic of control involves decision making based on factors that the entrepreneur can control as opposed to prediction of future events. As the iterative process of effectuation evolves, the entrepreneur accumulates new means and goals, and converges to a set of effects resulting in an artifact that embodies the desired aspirations and goals.

The theory of effectuation has been employed in the entrepreneurship domain to explain generation of entrepreneurial opportunities (Sarasvathy et al. 2003), entrepreneurial behavior (Fisher 2012), decision making under uncertainty (Wiltbank et al. 2006), and new venture development (Sarasvathy 2001). Prior literature has favored qualitative data analysis using case studies for empirical analysis. This has been due to the suitability of qualitative data and case study to understand the phenomenon of interest and lack of quantitative measurements for effectuation constructs.
The theory of effectuation has recently been employed in information systems literature. Drechsler and Hevner (2015) provide guidance for incorporating the concepts of effectuation into the design science research (DSR) paradigm. They argue that effectuation-oriented DSR may provide superior lens to examine problem spaces that are characterized with uncertainty and dynamic evolution. In this dissertation, we take this conceptualization one step further to propose an effectual process to develop novel applications on software-based platforms. In the next section, we discuss the underlying logic of existing software development approaches and identify the strong grounds of an effectual approach using the framework of prediction and control.

2.5 Prediction vs Control in Software Development

The proposed effectual software development process can be contrasted to more traditional approaches for developing software such as plan-driven, controlled-flexible, and ad-hoc (Harris et al. 2009a) by considering (Figure 4) the dimensions of control (x-axis) and prediction (y-axis) (Wiltbank et al. 2006). Increasing prediction posits that a development organization can predict the exogenous environment and position itself to be relevant in the future via planning. Increasing control focuses on the ability of an organization to control and shape its own endogenous environment to be relevant in the future via adaptation.

The planning approach advocates predicting the exogenous environment and positioning accordingly. Predicting the exogeneous environment requires analysis of different factors that are known to influence the environment, analyzing trends, and evaluating alternative strategies that may be best going forward. As uncertainty increases, planning approach advocates analyzing additional information to reduce uncertainty.
A predominately adaptive approach suggests positioning for future relevance by rapid adaptation. In comparison to planning approach, an adaptive approach does not emphasize analysis of alternatives and information gathering to reduce uncertainty. Instead, adaptive strategy emphasizes flexibility to manage uncertainty. Following flexibility, adaptive approach emphasizes feedback from the environment to gauge the appropriateness of its actions—will the firm be relevant in this position? If not, it adapts to stay relevant. Some firms bridge the planning and adaptive approach by planning to adapt. In such an approach, the firm quickly identifies and analyzes alternatives and repositions based on feedback from its environment. Such fast decision-making strategy allows the firm to retain strategic planning and flexibility to adapt.

With increasing emphasis on control, the strategy involves actively shaping the environment by making it endogenous rather than navigating and positioning in an exogenous environment. In

---

**Figure 4. Framework of Prediction and Control (Wiltbank et al. 2006)**

A predominately adaptive approach suggests positioning for future relevance by rapid adaptation. In comparison to planning approach, an adaptive approach does not emphasize analysis of alternatives and information gathering to reduce uncertainty. Instead, adaptive strategy emphasizes flexibility to manage uncertainty. Following flexibility, adaptive approach emphasizes feedback from the environment to gauge the appropriateness of its actions—will the firm be relevant in this position? If not, it adapts to stay relevant. Some firms bridge the planning and adaptive approach by planning to adapt. In such an approach, the firm quickly identifies and analyzes alternatives and repositions based on feedback from its environment. Such fast decision-making strategy allows the firm to retain strategic planning and flexibility to adapt.

With increasing emphasis on control, the strategy involves actively shaping the environment by making it endogenous rather than navigating and positioning in an exogenous environment. In
the *visionary* strategy, the firm emphasizes construction by considering possible alternatives and proactively working to realize their potential. Consideration and analysis of possible alternatives includes predicting the future possibilities and the alternatives’ potential to be relevant. However, proactively working to realize the alternatives includes treating the environment as endogenous and achieving goals by gathering required resources.

![Software development approaches and the framework of prediction and control](image)

**Figure 5. Software development approaches and the framework of prediction and control**

A transformative strategy focuses on controlling its environment. However, unlike visionary approach, transformative approach does not emphasize consideration and analysis of alternatives. Instead, transformative strategy focuses on existing means to derive possible alternatives and select alternatives which allow the firm to embrace future contingencies. Focusing on existing means, the transformative approaches can improve their goals and means with intermediate artifacts. Together, these approaches show the dominant strategies followed by firms to balance uncertainty.
Applying this thinking to software development (Figure 5), the planning approach defines the underlying logic of a plan-driven development approach. In plan-driven software development approach, project plans are devised, and resources are identified and acquired, at the start, with an understanding that the software product will be relevant (and profitable) upon its completion. With few controls in place (i.e. trial and error), an adaptive approach leads to an ad-hoc approach to software development, where the development team is constantly calibrating its course by reacting to changes in the environment. Bridging the planning and adaptive approach is the controlled-flexible approach in software development, where planned control mechanisms are inherently prediction-based while emergent control mechanisms introduce flexibility to adapt to uncertain environments (Harris et al. 2009a). Agile development methods are typically of this variety, combining some up-front planning with dynamic controls during the execution of the development project.

2.6 Effectual Software Development

A relatively unexplored area in the matrix of Figure 5 is the transformative quadrant where the exogenous market and environment are highly unpredictable. The development organization must rely on endogenous dynamic controls to design and develop novel, profitable products and services. This is the transformative space in which effectuation theory can be applied (Wiltbank et al. 2006). Based on the existing resources at hand, possible alternative actions (i.e. effects) are evaluated, and those alternatives are pursued that best shape the endogenous environment. As the organization moves through this transformative approach, new resources and aspirations are identified and refined until a final innovative result is achieved (Sarasvathy 2001). Contrast this to a goal driven approach, where the innovative outcome has to be known a priori, and the pursuit is to realize the goal. We are not aware of any software development approach that
embodies such an underlying logic that emphasizes existing resources and aspirations as the starting point and focuses on the controllable aspects of the future.

Software development on digital platforms provide uncertain, resource constrained, and high-risk environments. Such a setting renders software development approaches from the traditional realm of the high prediction quadrants - planning and visionary - infeasible for application development. Drawing from the theory of effectuation, we propose an effectual software development process that supports such a transformative approach for developing novel applications on digital platforms.

The development of novel software applications requires creative design activities. Drechsler and Hevner (2015) provide guidance for incorporating the concepts of effectuation into the design science research (DSR) paradigm. They argue that effectuation-oriented DSR may provide a superior lens to examine problem spaces that are characterized with uncertainty and dynamic evolution. In our research, we take this conceptualization one step further to propose an effectual process to develop novel applications on software-based platforms. Effectual thinking aligns with software development on digital platforms due to the limitations of causal based approaches in the literature (Harris et al. 2009a) and the challenges offered by digital platforms. Causation based approaches to software development identify a goal and realize it through a linear and/or iterative process. These are prediction-based approaches, where the application’s ultimate fit and utility in the platform context is identified a priori (Gill and Hevner 2013). Such a priori identification of application’s utility is possible in environments that are characterized by certainty and stability.

However, software development on digital platforms must navigate uncertain, resource constrained, and high-risk environments. Such settings render software development approaches
from the traditional realm of prediction infeasible for application development. The central
tension in software development is achieving a balance between planning and flexibility.
Controlled-flexible development processes balance planning and flexibility under increasing
uncertainty in market and technology. The initial project landscape provides a partial
specification of the product to be developed. Adapting to environmental uncertainty, feedback
mechanisms allow the specification to be modified so that the product can match changing
market needs. Controlling the solution space, scope boundaries allow exploration of solution by
the project team while constraining their space to align with organizational goals. Thus, iterative
development, scope boundaries, and feedback mechanisms allow the software product to be
relevant upon completion (Harris et al. 2009a). Though a controlled-flexible approach develops a
relevant product in uncertain environment, none of the prior literature has explored its
applicability to the development of novel software products on digital platforms.

We contend that current agile development processes and methods do not effectively explain
and address the challenges of digital platforms because (a) any form of prediction in highly
dynamic environments is suspect, (b) fixed development constraints on platforms are not
conducive to agile thinking, (c) risk and loss tolerance are key factors in whether or not to build
an application in a risky environment and are not highlighted in agile processes, and (d) the
platform development process must go beyond product-market match to consider the factors of
product-platform match, value add beyond the core platform values, and novelty of the offering.
Consideration of these issues requires a software development approach that treats the uncertain
environment as endogenous and shapes it (Transformative quadrant in Figure 5). APPENDIX 1
discusses key differentiating aspects of plan-driven, ad-hoc, controlled-flexible, and effectual
approach.
Digital platforms represent socio-technical, dynamic, and challenging contexts for software development teams. Using effectual thinking allows software development teams to identify multiple possible effects based on their available means. Through market and stakeholder feedback, the development team can iteratively attenuate their aspirations and identify an appropriate effect that embodies their aspirations, fits the application context, and provides utility to its users. This approach is in contrast to the causal approach since the team does not identify a particular goal; rather they iteratively attenuate their aspirations to arrive at the desired effect (artifact).

2.7 Summary

Software platforms provide unique challenges to software application development teams which have not been considered in prior literature. To maintain user base and relevance, application developers and platforms need novel applications on the platform. The framework of prediction and control helps us to understand the underlying logic of existing approaches to software development and highlight their limited applicability to risky and resource-constrained environment of digital platforms. The theory of effectuation provides an avenue to develop an effectual approach to software development by shaping the environment. The effectual approach emphasizes control over prediction by identifying alternatives based on existing means. An effectual approach to software development holds promise to extend current software development approaches by incorporating effectual thinking.
CHAPTER 3. AN EFFECTUAL APPROACH TO SOFTWARE DEVELOPMENT

The theory of effectuation focuses on controlling the environment rather than positioning in the environment. To control its environment, the theory advances that intermediate effects are identified based on existing means, acceptable risk, aspirations, and controllable aspects of the team. Intermediate effects develop feedback loops which increase the team’s resources and attenuate aspirations.

To understand the effectual software development approach, we start by mapping some of the key ideas in the theory of effectuation with software development. In this approach, we view the software development team as entrepreneurs. While entrepreneurs can identify different domains to develop its enterprise, the software development team is usually constrained by the domain and type of application they will develop. Thus, software development teams need to consider the dynamics associated with the context in which they are operating.

Similar to entrepreneurs, software development teams can draw upon their means which include technological and domain knowledge. Also, the software development team can identify its aspirations for the software application. As discussed earlier, the application’s context may play an important role to identify intermediate effects for the development team. Intermediate effects may include nightly or weekly builds which allow the software development team to identify new resources and attenuate their aspirations. After acceptable match of aspirations and an intermediate effect, the team may deliver the application to its users.
3.1 A Model of the Effectual Software Development Process

![Diagram of the Effectual Development Process]

Figure 6. Components and Relationships of the Effectual Development Process

The evaluation of the evolving effectual application development process begins with a fuller understanding of components of the process and their relationships as presented in Figure 6. The three key components of the process model are:

- **Means** for the project manager and development team are the existing resources that are available to them. It consists of technology and skills (programming language, API’s, tools), market knowledge (customer orientation, seasonal trends, patterns from archival data), platform knowledge (connection interface, tools and technology, best practices, available API’s on the platform), control mechanisms (scope boundaries, stakeholder feedback), and the social capital that they can draw upon.

- The **software platform** provides a set of resources and constraints. For example, the connection interfaces to the platform, development guidelines, tutorials, and development
standards that provide resources for the project to draw upon while constraining them to those specific alternatives.

- Four aspirations for the project team are identified – application-platform match, application-market match, value proposition of the application should exceed the core value proposition of the platform, and novelty of the application.

For the development team, means, platform, and aspirations exist a priori to the development process. Drawing on its means and aspirations, the software development team lists action alternatives that can be undertaken. An action may encompass identification of new application feature, fixing existing issue, or removal of existing features from the application. Identification of actions draws on a subset of means and aspirations. Thus, an identified action may draw on technological means to satisfy application-platform match while another action may draw on market and platform knowledge to accomplish novelty. Pervasive in the identification of actions is the platform’s capabilities and constraints that the team must consider.

The mechanism to select appropriate actions from identified action alternatives is provided by two heuristics: acceptable risk and logic of control. According to classical decision theory, risk associated with an alternative is the variation in its possible outcomes (March and Shapira 1987). The larger the variation in possible outcomes, the larger is the risk associated with the alternative. Thus, evaluation of potential actions is based on the trade-off between its expected return and associated risk. An action is said to possess acceptable risk if the development team can perform corrective actions in case the alternative does not satisfy the team’s aspirations.

Further, the managerial perspective notes that risk is controllable and modifiable through skills and information (MacCrimmon and Wehrung 1986). The logic of control emphasizes
controllable aspects of future events i.e., a focus on aspirations that can be controlled by the project team (Sarasvathy 2001). This translates to favoring actions that can be controlled by the team given its means and aspirations. For example, implementation of a feature is not favored if it requires API’s from the platform which are not yet available. The project team conducts a risk analysis (Benaroch et al. 2006; Flyvbjerg and Budzier 2011; Lyytinen et al. 1998) on the set of possible actions. Actions that have an acceptable risk are identified. Platform state, existing portfolio of controls (Harris et al. 2009; Kirsch 1997), and aspirations of the project team identify the controllable aspects of the possible actions.

Together, actions selected through this mechanism give rise to an effect. An effect is the operationalization of abstract aspirations (Sarasvathy 2001). Specifically, an effect encompasses all the features and operational specifications of the application that can be developed by the team. In software engineering terminology, an effect represents the software specification. Similarly, an effect may include intermediate deliverables (nightly or weekly builds) which may be demoed to users and/or clients. With effectual cycle, the software development should grow the system artifact which represents the knowledge base of the team. The team knowledge base evaluates its aspirations with this knowledge base and identifies new resources. This mechanism gives rise to new means and constraints for the development team – expanding cycle of resources. New means stem from an improved understanding of the problem space. Similarly, new constraints are identified that help retain appropriate and promising aspects of the aspiration – converging cycle of constraints. Finally, the Artifact (application product or service) is the realization of team’s aspirations and is developed/implemented by the team.

To validate the ideas and research model presented in this section, we perform exploratory studies. First, we perform a qualitative study with secondary data of an open source
application. Second, we conduct pilot interviews at local companies. These studies allow us to address construct, internal, and external validity of the research model. In what follows, we discuss the studies.

3.2 Preliminary Evidence of Effectual Thinking in Platform Applications

To investigate (initial confirmation of effectual thinking as found in existing software development projects on digital platforms) the new ideas of effectual software development on digital platforms, we perform a qualitative study of open-source application development projects. Three key selection criteria are established to identify appropriate samples for data collection and analysis: (a) the application should not be developed by an individual only, (b) the digital platform should be owned by a different organization, and (c) application users should have alternative options other than the application under investigation (Malgonde and Hevner 2017). We identified Apache Cordova as an open-source mobile application development framework (Figure 7). The application is developed by a distributed team of contributors. The digital platform on which the application is built is controlled by disparate organizations. Finally, rival applications for Apache Cordova are available to its users.

3.2.1 Apache Cordova

Following the mantra of Apache Software Foundation (ASF), the Cordova application framework is used by numerous application developers to develop applications and provides tools and interfaces that can be readily used by developers. Apache Cordova provides all the interfaces and plugins that the development team needs to develop an application which can then be published across multiple platforms. Cordova supports seven platforms—Android, iOS, Windows, Ubuntu, Blackberry 10, WP8, and OS X. Web View provides user interface capabilities, Web App provides configurational settings for the application, and Cordova Plugins
allow seamless communication within application components and the platform. The *Mobile OS* platform provides standardized plugins, which are regularly updated by the platform owner.

![Apache Cordova Architecture](https://cordova.apache.org/docs/en/latest/guide/overview/)

**Figure 7. Apache Cordova Architecture**

All Apache projects are required to store and host programming activities, decisions, and status of the project. Projects adhere to these requirements using mailing lists, project management and version control tools, and/or messaging platforms. In our study, we extract data from the project management tool. Specifically, we focus on this dataset because (a) all data are available, (b) the dataset consists of issues raised by active contributors, and (c) the dataset includes requests for information, bug fixes, feature requests, suggestions, and discussions. We focus on completed *user stories* that describe a specific feature request and/or issue with the application and/or platform. Completed user stories are suitable for this research since they

---

provide the issue and its description addressed in the story, and a solution that is provided and implemented in the application. Some stories have additional discussions on the viability of alternative solutions to the issue being discussed. Story descriptions and related comments for over 1,000 stories were extracted and analyzed. The data analysis is supported with documents from proposals, board reports, and project documentation.

### 3.2.2 Analysis of Apache Cordova Stories

The author of this study performs analysis of the data as follows. First, inspecting all stories in the database, we remove unclear or non-descriptive stories. These include stories that do not discuss any specific issue in depth, provide a link or non-conclusive short description, and/or provide a blob of program code without accompanying discussion. The user story needs to clearly present the issue at hand. As the initial inspection retained clear and descriptive stories, they were subjected to qualitative analyses. These analyses include coding the data with identification of relevant terms and definitions. Finally, inferences were derived from selected stories and triangulated from multiple sources. Through these rigorous filters, we refined the initial set of 1,000 stories in order to identify 42 user stories with sufficient detail for full analysis. We use Atlas.ti qualitative data analysis software for our analysis. To aid our coding procedure, we developed a qualitative codebook that identifies sub-codes and operational definitions (Table 1) for each construct in our model.
<table>
<thead>
<tr>
<th>Construct</th>
<th>First Cycle Code</th>
<th>Operational Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Means (existing resources at hand)</strong></td>
<td>Technology</td>
<td>Existing technological capability within the team (in this case, the community) – programming languages, tools, configuration, testing, documentation, etc.</td>
</tr>
<tr>
<td></td>
<td>Market knowledge</td>
<td>Existing knowledge about the platform market (alternatives, competitors)</td>
</tr>
<tr>
<td></td>
<td>Platform knowledge</td>
<td>Existing knowledge about the technological state of the platform</td>
</tr>
<tr>
<td></td>
<td>Social Capital</td>
<td>Capital that the team can draw upon to append existing means</td>
</tr>
<tr>
<td><strong>Platform (resources and constraints provided by the platform)</strong></td>
<td>Technology (API)</td>
<td>Technological resources and constraints provided by the platform (APIs, programming language, setup, features)</td>
</tr>
<tr>
<td></td>
<td>Market</td>
<td>Existing offerings on the platform market</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Existing value offered by the platform to its customer (in terms of features that the users can use – tangible)</td>
</tr>
<tr>
<td><strong>Aspirations</strong></td>
<td>Product-market match</td>
<td>The features to be built in the product should match the requirement of the market</td>
</tr>
<tr>
<td></td>
<td>Product-platform match</td>
<td>The product should be technologically compatible and functional on the platform</td>
</tr>
<tr>
<td></td>
<td>Exceed Platform Value</td>
<td>The features being built in the application should help exceed the application the core set of value provided by the platform</td>
</tr>
<tr>
<td></td>
<td>Novelty</td>
<td>Technological or feature based novelty of the application that the existing applications and platform do not cover.</td>
</tr>
<tr>
<td><strong>Acceptable Risk</strong></td>
<td>Commit limited resource</td>
<td>Commit limited technological and people resources to any given feature.</td>
</tr>
<tr>
<td></td>
<td>Application recoverable after failure</td>
<td>If implementation of the given feature results in failure, it should not jeopardize entire application.</td>
</tr>
<tr>
<td></td>
<td>Risk Analysis</td>
<td>Risk portfolio of an alternative are determined before decision-making.</td>
</tr>
<tr>
<td><strong>Logic of Control</strong></td>
<td>Logic of Control</td>
<td>Decision making based on factors that the team can control as opposed to prediction of future events.</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Fixed bugs</td>
<td>The issues that were identified based on means and fixed.</td>
</tr>
<tr>
<td></td>
<td>Completed Tasks</td>
<td>Feature requests which were identified and completed using means and acceptable risk.</td>
</tr>
<tr>
<td><strong>Effects</strong></td>
<td>NA</td>
<td>Collective documentation and understanding of which features and issues are to be addressed in the project.</td>
</tr>
<tr>
<td><strong>Expanding Cycle of Resources</strong></td>
<td>New technological knowledge</td>
<td>Identify new API’s, tools, and configurations that can be used by the application.</td>
</tr>
<tr>
<td></td>
<td>New market knowledge</td>
<td>Identify new requirements that the market needs.</td>
</tr>
<tr>
<td></td>
<td>New platform knowledge</td>
<td>Identify new API’s, tools, and configurations that are provided by the platform.</td>
</tr>
<tr>
<td><strong>Converging Cycle of Constraints</strong></td>
<td>Converging technological (means) constraints</td>
<td>Identify specific API, tool, or configuration for the application from competing alternatives.</td>
</tr>
<tr>
<td></td>
<td>Converging feature constraints</td>
<td>Identify specific feature for the application from competing alternatives.</td>
</tr>
<tr>
<td></td>
<td>Converging platform constraints</td>
<td>Identify specific API, tool, feature, or configurations competing alternatives provided by the platform.</td>
</tr>
</tbody>
</table>
The sub-codes are identified from the research context, theoretical constructs, conceptual framework, and research question. Operational definitions are identified based on the research context and prior empirical studies on effectuation (Chandler et al. 2011, Perry et al. 2012). Further, the coding scheme is flexible to add new sub-codes as they emerge from the data and update the operational definitions. The codebook guided our first-order coding. Using descriptive coding technique (Miles et al. 2013), sub-codes from the codebook were applied to each story where applicable. APPENDIX 2 provides several sample stories from our database and the codes that are assigned to them. To address construct validity, multiple sources of data—stories, documentation, contributor comments, board reports, and proposals—are tapped to ensure that the findings converge. Reliability of the study is addressed with (a) programmatically retrieving and storing analyzed stories locally from the project management tool, (b) maintaining the qualitative codebook of codes and operational definitions, and (c) developing matrices from the labelled data.

3.2.3 Findings from Apache Cordova Stories

The results of the qualitative analyses of the Apache Cordova projects are presented in Table 2 including the first cycle codes (and related constructs identified in Figure 6) and the frequency of the codes. As the secondary data used for this analysis consist of contributors’ descriptions of issues and feature requests for the Cordova applications, the data are characteristically technical in nature. This readily translates into identification of technological means available to the application development team that is specific to the application and platform. We identified 40 stories that show technological means for the development team. Available means include knowledge about market needs (feature requests), value propositions provided by the platforms, and new features that are introduced by platforms or competing applications (through developer
conferences or official press releases). Similarly, the technological opportunities and limitations by platforms are discussed by contributors. Current working of API’s and the value they provide to the user are discussed and coded in 23 stories. This leads to identification of limitations and opportunities that serve as value additions to the current value proposition of the platform and serve the market need. This evidence points to the nascent market in which the project focuses. Building on existing technological and market knowledge, possible alternatives are identified. Further, these stories do not predict potential changes to the market and platform. Instead, the focus is to build the application based on current understanding of the technology, platform, and market.

The analysis also leads to identification of aspirations in the team’s decision making and actions. Specifically, the application-platform match is one of the central driving forces across these stories since contributors focus on technical aspects that lead to seamless operation between the application and platform. 24 stories are coded to identify application-platform match. Further, the analysis finds support for the aspiration of introducing novelty to the application (15 stories) and ultimately adding value to the existing value proposition provided by the platforms (14 stories). The common theme in these aspirations is identification of opportunities (limitations and/or enhancements) for value addition through existing means and platform knowledge and introducing novel features that take advantage of the platform’s opportunities.
Table 2. Constructs and their Frequency in the Apache Cordova Data

<table>
<thead>
<tr>
<th>Construct</th>
<th>First Cycle Code</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Means</td>
<td>Technology</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Market knowledge</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Platform knowledge</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Social Capital</td>
<td>2</td>
</tr>
<tr>
<td>Platform</td>
<td>Technology</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Market</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>8</td>
</tr>
<tr>
<td>Aspirations</td>
<td>Product-market match</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Product-platform match</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Exceed Platform Value</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Novelty</td>
<td>15</td>
</tr>
<tr>
<td>Acceptable Risk</td>
<td>Commit limited resource</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Application recoverable after failure</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Risk Analysis</td>
<td>21</td>
</tr>
<tr>
<td>Logic of Control</td>
<td>Logic of Control</td>
<td>32</td>
</tr>
<tr>
<td>Action</td>
<td>Fixed bugs</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Completed Tasks</td>
<td>11</td>
</tr>
<tr>
<td>Expanding Cycle of Resources</td>
<td>New technological knowledge</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>New market knowledge</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>New platform knowledge</td>
<td>21</td>
</tr>
<tr>
<td>Converging Cycle of Constraints</td>
<td>Converging technological constraints</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Converging feature constraints</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Converging platform constraints</td>
<td>11</td>
</tr>
</tbody>
</table>
The heuristics of acceptable risk and logic of control also find strong support in our analysis. Each story is identified and addressed by (typically) one contributor. Thus, the team is devoting limited resources for each issue and feature, and 33 stories are coded for this sub-code. Alternatives identified—do feature A or B or C—accompany risk analyses that discuss technological implications on the application and platform, novelty, and extending the platform’s value proposition. 21 stories are coded to show risk analysis and identify alternatives that have acceptable risk associated with them. Further, actions identified by the team embody the logic of control and are coded in 32 stories. These include decisions based on the current means, platform knowledge, and the aspirations of the team, rather than predicting which actions would enhance the application. Finally, the application is already in use by an array of users which provide feedback to the development team. This represents a control driven approach rather than prediction-based approach that would identify the goals of an application a priori.

Actions (32 stories coded) lead to intermediate effects, which are the operationalization of team aspirations. Each iteration of the Cordova application served as an intermediate effect that, in turn, expanded means and attenuated aspirations. Specifically, intermediate effects help identify technological avenues, tools, limitations, and features, that increase the fit and utility of the artifact. 37 stories are coded to identify expanding technological knowledge. In addition, intermediate effects improve the platform knowledge for the overall team, as new features are implemented that connect to the platform and add new value to its existing value proposition.

Overall, the frequency of sub-codes identified in our analyses justifies the conjecture that software development teams developing novel applications on digital platforms employ the constructs of effectual thinking even when the terms used in the processes may not align with those used in effectuation context. Also, these stories span across multiple iterations. For every
iteration, a set of stories represent the intermediate effect that is developed and tested. This provides feedback to the development team that expands its resources and attenuates its aspirations. APPENDIX 3 presents and discusses several illustrative exemplars of this qualitative study.

The focus of this qualitative data analyses is to identify evidence that supports the current (perhaps, unconscious) use of effectual thinking in open-source development projects in the Apache Cordova environment. To address these limitations, we developed operational definitions for effectuation constructs in the software development context and updated them as the data analyses progressed. Also, stories selected for analyses provided extended discussion on the issue at hand. Based on these analyses, we did find considerable evidence that demonstrates the wide-spread use of effectual thinking in the Apache Cordova projects. A limitation of this study is that our data analysis is limited to qualitative secondary data for available open source projects. Specifically, the software development projects studied did not use effectual concepts and terms directly. Thus, the user stories required subjective coding and interpretation via an effectual lens. To address this limitation, we conduct another qualitative study with primary data from pilot interviews.

3.3 Qualitative Study with Pilot Interviews

The goal of this pilot study is threefold. First, this study aims to validate and augment the research model. Second, this study will develop the interview protocol for a broader qualitative study. The interview protocol is based on the operational definitions developed in the preliminary study. Third, this study also aims to validate the data analysis procedure which will be used in later study to validate the research model.
3.3.1 Interview Design and Protocol

Based on the operational definitions of constructs in the effectual model, we develop an interview protocol. The interview protocol for project manager and team members is provided in APPENDIX 4. The interview protocol includes introductory questions which focus on the application and its domain, the platform on which the application is developed, interviewee’s role in the project, and the general process followed in the project. Following such introductory questions, the protocol focuses on the novelty of the application as perceived by the interviewee. Following the discussion on the novelty of the application and critical success criteria for the application, the interview protocol includes questions pertaining to model’s constructs, and concludes with a discussion of the research model.

3.3.2 Pilot Interviews

To identify potential projects for pilot interviews, our selection criteria includes software application development projects that are developing novel applications on digital platforms. We are open to different application domains. However, we require the project to be developed by team of software development professionals rather than individual entrepreneurs or ad-hoc developers. Two project managers from local organizations were identified for pilot interviews.

The local organizations and project represented by the project managers are:

- An IT-department of a non-profit educational organization that is developing a novel application that serves universities reporting obligations to state-mandated or requested entities. We label the project manager from this organization as PM1. The interview with PM1 lasted about 32 minutes and resulted in 12 pages (69 passages) of transcript.
- A Fortune-500 organization with a development team exceeding ten and a project manager. This team is developing an application that supports online subscription of
enterprise software. The application connects to multiple external platforms, increasing the complexity of the task. Interestingly, the application serves as a platform to other connected applications. We label the project manager from this organization as PM2. Interview with PM2 lasted about 49 minutes and resulted in 17 pages (78 passages) of transcript.

3.3.3 Data Analysis

Transcribed interviews were analyzed by the author and a Professor at USF. Independent open coding is performed on the two interview transcripts. Upon completion, coding was consolidated to identify agreements, disagreements, conceptual ambiguity, and disconnect between operational definitions and conceptual constructs. Second-level coding is not performed.

3.3.4 Results

The pilot interviews and their analyses help us to validate and revise our interview procedure for a primary study. Specifically, we validate the set of interview questions, their order, and the possible avenues to expand during the next set of interviews. Further, we revised our interview protocol based on our experience during the pilot interviews. APPENDIX 5 provides the revised interview protocol. The analysis of pilot interviews validates the feasibility of coding the data based on the operational definitions of the constructs and subconstructs.

Common across the two pilot interview teams is the alignment with controlled-flexible approach (Harris et al. 2009a). The teams identify 2-4 week sprints as their development schedule to deliver functionality. At the end of sprint, they receive feedback from application users that is incorporated in later sprints.

We find that market uncertainty plays an important role in these applications. Harris et al. (2009a) identify technological uncertainty as another contributing factor to the choice of
application development approach. However, platform-based application development is typically limited to certain tools and technology supported by the platform.

The platform’s state also plays an important role in the applications development cycle. The digital platform enables the application development by offering newer APIs, management tools, and documentation. Simultaneously, the platform constrains the application development team by deprecating APIs and release updates to existing platform components. We also note that the platform’s motivation to update its components is stemmed by competing platform ecosystems, technological advancement, and generativity.

The software application development teams in the pilot interviews relied on mockups and designs to communicate ideas. PM1 specifically discusses the rapid prototyping capability provided by the platform. Such visual artifacts allow the team to swiftly resolve conflicts and respond to market and platform changes.

Finally, we note the increasing focus on discussions and proof of concepts to new ideas. As new ideas and features are provided to or originate within the team, there persists substantial ambiguity about the downstream impact of incorporating them on profitability, acceptability, and performance of the application. Discussion focusing on the architectural, design, and implementation feasibility of such requests is highlighted in PM2’s team. Similarly, the willingness and ability of the team to develop, evaluate, and revise, proof of concepts for these debated ideas is mature.

3.4 Revised Model of the Effectual Software Development Approach

The qualitative study with secondary data and pilot interviews provide preliminary evidence on the presence of effectual thinking in software development projects for novel applications and
validate the interview protocol for follow up study to validate the research model. In addition to these expected outcomes, these studies also provide findings to augment the research model presented in prior subsection. In this subsection, we discuss these findings from the two studies and develop the revised model of effectual software development approach (Figure 8).

**Figure 8. Revised Model of Effectual Software Development Process**

3.4.1 **Release Application to Platform**

In the earlier model of effectual software development process (Figure 6), the outcome of the effectual process is the artifact (software application). Once the final artifact has been delivered, the process is assumed to be complete. In this sub-subsection, we focus on two areas of the model that will be revised based on the studies discussed in prior subsections: (a) the effectual process is assumed to be complete, once the artifact is completed, and (b) the process model is less forthcoming on the decision criteria to exit the effectuation process because the model assumes existence of an exit point.
In the secondary data analysis for Apache Cordova project, we find that the Apache Cordova application has numerous ‘releases’ to the platform. These releases represent outcomes of effectual process. Each release may include minor and/or major improvements, bug fixes, and enhancements. For example, one of the stories in our database was discussing an issue which could be fixed in the upcoming release or moved to next release.

“I marked this initially as a blocker for the Callback 2.0.0 release, but as discussed on dev@ we may perhaps want to move this to a separate WEINRE issue tracker and target for the separate initial Apache release of weinre.”

This shows a common understanding in the team that the next release of the software application is not the last or concluding release for the application. We see three primary reasons for this continual need for release. First, the application development team prioritizes issues based on contextual needs. Consider the previous story description from our database. The team prioritizes issues which demand immediate attention. Consider another example from our pilot interview with PM2:

“[the team] would estimate during sprint planning for each user-story based on the priority that I gave them. And also of the technical feasibility, we would reorder it and then they would start estimating each story. Then when we reach 40, we would stop and that’s when the sprint would actually kick off”

Second, we see a market-driven need for frequent release to the platform. As competitors update their application and users’ requirements evolve, the application needs to be updated in accordance with these market forces. For example, PM2 talks about the importance of ‘keeping-up’ with the competition:
It's important because technology keeps changing, so our [competitors] keeps upgrading, that's the biggest keys source. They are in the Silicon Valley. They have newer products coming out and everything has to be developed and is done in a different way. If we don't match with them, we would not be able to satisfy or sell new key things. If we don't do that, we obviously don't sustain in the market and we will not be beside a lot of products.

Finally, changes to the digital platform require frequent release to ensure that the application is compatible with the platform. These changes are often in the form of changes to APIs—add, update, or deprecate. As a consequence, the development has to ensure that every release uses updated platform APIs. For example, in the Apache Cordova application, this story discusses new dependencies with Apple’s iOS update:

The ALAssetsLibrary framework has been deprecated in iOS 9, replaced by the Photos.framework. Once our minimum dependency is iOS 9, move to it. Usage:


Update deprecated ALAssetsLibrary usage in plugins

Related to the issue of sequential release of the application is the decision of releasing the application. In the current model of effectual software development process, there exist an exit point which leads to the final artifact. However, with the newer understanding of continual releases, we see that the decision-making point has different contextual characteristics associated with it. For example, PM2 notes that the release was authorized upon user testing.

It will be an actual demonstration in our test environment. After that, we would release it out for testing for our business folks. After the business team had
reviewed it, we would push it to the production, Then if there are any bugs, and we would work on it in the consequent sprints and likewise.

However, issues were logged and prioritized for the next sprint. Also, the decision to release is influenced by other organizational issues such as change management, application data, or market requirements.

*I’m hesitant to say exactly two weeks is because it's a big enterprise and we have changed controls and changed phrases to why something is like a month in and things like that. In two weeks, we would have a production deployment ready to go but we would wait for all our internal procedures to be done.*

*Say, if it's a month in, we won't release and they would release right after to avoid any impact to our business teams. It would be ready for a production release but depending on the data situations it would be released to production.*

With this improved understanding of the development process, we revise the research model Figure 8 to include a decision point to identify a release for the software application. In addition to the decision point, the model is also revised to include feedback from the application’s release to the next process cycle.

### 3.4.2 Intermediate Effects

Release of the software application allows the application development team to better understand the state of the platform and provides feedback to subsequent iteration of the effectual process. With this new understanding of the application’s release, the intermediate effects identified by the model also needs revision. Particularly, we discussed *effects* as intermediate iteration outputs which helped the software application development team to expand its resources and attenuate aspirations.
The prior definition of effects was restrictive to workable code that can be delivered when the exit criteria was met. From the studies, we find that the effects are intermediate effects which include documentation, demos, nightly builds, design documents, user interface mockups, proof of concepts, backlog items, and so on. This includes an exhaustive list of intermediate artifacts which allow the team to identify new avenues (features, improvements, bugs) and attenuate aspirations. The effects are ‘intermediate’ because they represent artifacts which are in the state of development. Effects embody the current understanding of the software application development team. Effects also serve as a point of reference to validate ideas and decisions. For example, PM2 discusses extensive use of proof of concepts to identify and vet ideas for the application.

We do as little proof of concept to see how valid it is. API, just opinion C, what values are returning or connect and see a few systems, pull data to see how it works. We use that as a user story. It's a proof of concept user story so that people work on it. And then, we define how we proceed from there.

Similarly, PM1 discusses intermediate artifacts of mockups to illustrate the ideas. In this example, mockups help the users to identify and attenuate aspects of their aspirations. At the same time, mockups allow the development team to identify components required to realize the feature. Also, the mockups help the team to share ideas, identify improvements in their current understanding of the feature, and identify the feature’s aspects that will be incorporated in the application.

What happens is the users will just say that this is on their wish list and development team might sometimes just say that even though you are thinking that's complicated it's actually not. They can just say it's a two-day thing. That can be done very easily. What happens is they will just present a mock-up of
An important effect in the development process is the backlog. Backlog is the central repository of stories identified by the team. Each backlog item (story) consists of description, priority, and team’s notes associated with the story. During iterations, the team revise their backlogs to represent new understanding of the application and its environment. These changes may include changing priority of stories and descriptions to add and/or edit descriptions. Thus, the backlog is an important intermediate effect for the team.

3.4.3 Identify new Subconstructs

The model of effectual software development process identified four subconstructs for the construct of *means*—technology, market knowledge, platform knowledge, and social capital. These subconstruct identify means of the team which are typically outward facing and built over time. In addition to these subconstructs, our studies identified the team’s and organization’s *culture* as another subconstruct to means.

Culture of the software development team represent the stated and unstated norms followed in the team. These norms may take the form of communication channels, hierarchical rigidity, and coordination mechanisms, among others. Some teams may prefer a formal mode of communication. For example, PM1 discusses the formal setup between application’s sponsors and the development team. The application’s sponsors requested features which were incorporated in the application. Contrast this to the case with PM2 where the team undertake ‘discussions’ to brainstorm ideas.

*A lot of these come in from the product managers and our business teams. Sometimes our development team as well because they explore or seen*
something and there's like, "Oh, this is something we can do. Are you guys interested?" There are multiple sources of inputs, or an executive who's looked at something and was like, "Oh, maybe if you provide this, it'd might be much more insightful."

It's a lot and all of it just streams down to us as product managers where we help get these things in place. Or discussions that we have with our [users], if we're designing a flow of how things are supposed to work, even that sometimes strikes gold.

Similarly, the Apache Cordova’s team showcased a culture where different individuals were allowed to post feature requests, submit solutions to existing bugs, and recommend suggestions, among others. For example, this story identifies and provides fix to an issue:

Application created with Cordova CLI using this command “cordova app build windows”. Windows 10 Universal App deployed in a Dell Machine with unplugged keyboard. Using a mouse everything works fine, but with unplugged keyboard in tablet mode: 1. Don’t shoe keyboard when text input focused. 2. Gestures like, drag, swipe doesn’t work.

In addition to the team’s culture, the organization’s culture also identifies means for the development team. For example, PM2 discussed the organization’s culture which influences the overall vision implemented by the team:

The SVP at [product name], [SVP’s name], he does not believe in us looking and chasing at what our competitors are doing. We focus on our customers and think of what they need versus what the competitors are doing. That's the key focus for us. So it happens, everybody has something different but the customer is what we want to focus on.
Prior studies also identify two subconstructs for the platform construct—maturity and complexity of the platform. Maturity of the platform represents the status of the platform’s technological, design, and governance mechanism of the platform. A platform is technologically mature if the connection interfaces provided by the platform are formally defined, documented, and perform well. A platform’s design allows the platform to include modular components and future changes to any components require minimal changes to dependent applications. Similarly, governance mechanisms outline the set of rules for platform participation. Together, a mature platform provides documentation on connection interfaces, follows release schedule, incorporates users’ feedback, and refreshes platform’s core offerings which can benefit users. In our studies, the platforms were mature. However, the type of context in which platform operate may influence the maturity of the platform. For example, the platforms involved in PM1 and PM2 were business facing. In other words, users of these platforms were businesses rather than individual users. Thus, changes to the platform were at limited pace and often accompanied documentation and tutorials. In case of Apache Cordova, the iOS and Android platforms were significantly fast-paced given the consumer facing nature.

In addition to the maturity of the platform, complexity of the platform also influences the application development team. Complexity of a platform is the platform’s ability to integrate and sync its components with other platforms. The components may include APIs, data, user interface, among others. PM2 discusses how their application interacts with multiple platforms:

*What [product name] does is we have multiple [platforms]. It could be [platform 1], [platform 2], [platform 3]. All of them have their subscription services. We create these in such a particular way that [users] in a market of--in a [industry name] can buy bulk of subscriptions or few subscriptions for each of the spender and the provisioning process that happens on the [platforms] as*
well as the [users] receiving it, and then the [users] transitioning this to the end-customers.

3.4.4 Nature of Application

Our earlier studies include applications which are diverse across multiple dimensions—size, interoperability, data, users, dependencies, technologies, among others. The diversity manifest as the applications operate across different markets, interact with different platforms, pursue different novel features. We find that the nature of application influences the overall process enacted by the software development team.

In case Apache Cordova, the application’s user base includes software developers who are developing their applications for iOS, Android, Windows, and other platforms. To support diverse components of these platforms, the Apache Cordova application’s size is high. In addition to the Cordova application, these developers (users) also use other tools to manage their data, processes, and components. In case of PM1’s application, the application was used by a set of individuals within the user’s organization. As far as the users were concerned, they did not connect or sync the application across multiple platforms. Consequently, the size of the application is small. In case of PM2’s application, the application’s size is high as the application connects to multiple platforms and users.

Another aspect of the application’s nature includes the data-intensity. The Apache Cordova application is less data intensive as it aims to develop the components required to develop applications. This implies a focus on developing modular components that can be updated to match different platforms. In contrast, PM1’s and PM2’s applications are data intensive. The data may reside along the application (PM1) or in cloud (PM2). This implies a focus on performance of the application.
Another interesting aspect of these applications’ nature is the status—for profit or non-profit. Apache Cordova and PM1’s applications are non-profit teams whereas PM2’s application is a for-profit team. While the three teams are driven to develop novel features and deliver value to its users, the focus on profitability of the application also plays a role in the development process.

The nature of application being developed influences the overall effectual software development process. For example, PM1 illustrates the process where limited size of the application dictates the features that the team will incorporate.

My role will be to understand the needs, understand the requirements, and at the same time be a person who might sometimes have to say no. That no, even though you think this is very good idea it will require a lot of resource and a lot of time and everything. It's not worth spending that much for the end game and at the same time, it's my job is to put something into action that was requested maybe by one person but will be used by many many others.

On the other hand, PM2 illustrates the process where feature requests are debated to identify ways in which they can be incorporated.

So if I break it down into saying someone comes up with the requirement which is the aspiration. That's where we would start saying this is what we aspire to sell or it matches the platform, anything, it's new, it adds value or it's a requirement, however you define it. We start there and then we start matching up to see, do we have the technology to support it? Do we have the money to support it?

Figure 8 illustrates the revised model of effectual software development process. The nature of application influences the overall process. Culture of the team and organization is
included as a subconstruct to the means available to the development team. Maturity and Complexity of the platform are included as subconstructs for Platform. Effects are intermediate artifacts which help the team to identify new resources and attenuate aspirations. The team decides if the intermediate artifact is mature to be released or deployed on the platform. The released artifact provides feedback to the team’s future effectuation cycles.

3.5 Summary

In this chapter, building on the theory of effectuation, we developed a process model of effectual software development process. The model identifies subconstructs for the constructs identified by the theory and explain the process. Preliminary support for the model is provided by a qualitative study using secondary data from Apache Cordova project which is developing a novel application. To validate the research model, a qualitative study is required. As a first step, we design the interview transcript and protocol. The interview protocol is validated and revised using a pilot study of two interviews with local companies. These studies led to the revised model of effectual approach to software development in Figure 8.

The qualitative analysis of secondary data and pilot interviews identify new constructs, subconstructs, and relationships in the model. Specifically, the *nature of application* is identified as a new construct which influences the effectual software development process. Given the size, data intensive nature, and interoperability needs of the application with other applications, different set of means and aspirations take center stage. Similarly, new subconstructs for means (culture) and platform (maturity and complexity) are identified that are specific to the platform context. In our earlier model, acceptable risk moderated the relationship between means and actions whereas logic of control moderated the relationship between aspirations and actions.
However, in our preliminary and pilot studies, we find that the acceptable risk and logic of control have a direct relationship to identify action alternatives which are pursued by the team.

Finally, our preliminary and pilot studies identify the effects (intermediate artifacts) identified by the theory of effectuation. Specifically, effects in software application development projects are design artifacts which may be in the form of proof of concepts, UI design, backlog items, and documentation. Further, the studies identify the decision point on deployment of the application to the platform or customer’s location. Upon deployment, the application initiates feedback to the next iteration of the software development process. In what follows, we discuss the design of study to validate the revised model of effectual software development.
CHAPTER 4. RESEARCH DESIGN

So far, this dissertation has identified unique software development challenges faced by software development teams in digital platform environments. These challenges extend the success criteria for the application to include application-platform match, application’s value exceeding platform’s value, and novelty of the application, in addition to the criteria of application-market match that has been explored in prior work. Based on the framework of control and prediction (Wiltbank et al. 2006), the study highlights limited applicability of existing software development approaches to the uncertain, resource-constrained, and risky environment of digital platforms and identifies the need for entrepreneurial thinking to achieve the success criteria. The theory of effectuation (Sarasvathy 2001) is identified as the theoretical basis to develop a research model of effectual software development which extends the current controlled-flexible approach in digital platform context. Finally, two studies were conducted to revise the model of effectual software development. As a confirmatory step, we now conduct two case studies to validate the revised research model and describe key characteristics of the effectual software development approach. Figure 9 illustrates the research design.
4.1 Study Design

We conduct case studies to validate the effectual software development approach. The case study methodology is appropriate when ‘how’ and ‘why’ questions are posed in the research (Yin 2008). In this study, we aim to understand how novel applications are developed in the context of digital platforms and how software development teams incorporate effectual thinking in their development processes. Specifically, focus on utilizing existing means to create
alternative effects, evaluation of effects based on aspirations, and use of decision heuristics to identify appropriate actions. Finally, the case study approach is appropriate since it allows a bounded yet rich exploration of the development of novel applications on software platforms. Case study methodology also allows us to extract a nuanced understanding of the effectual approach followed by software development teams. Further, case study methodology enables us to consider the actions, decisions, and heuristics, used by different team members. This helps us develop implications for different roles in the software development team.

We do not conduct a quantitative study for three reasons. First, the constructs identified in the theory of effectuation are difficult to measure (Perry et al. 2012). Although some studies have developed measures for the constructs (Chandler et al. 2011), the number of subconstructs in our research model make it infeasible to develop a questionnaire. Second, this study aims to propose, validate, and describe the effectual software development approach. This requires consideration of disparate views within the team and different processes followed by team members. Finally, prior studies building on the theory of effectuation have overwhelmingly used case study methodology. For a review of such studies, the reader is referred to Perry et al. (2012). Also, despite its strengths, a survey method would not be appropriate given the context and aim of this study.

4.2 Unit of Analysis

The unit of analysis for this study is the software development project developing novel application on digital platform. Also, the case studies include embedded units of analysis – the project manager and project team members. Interview questionnaires and data gathering focus on project manager and team members to identify inclusion of effectual decision alternatives and
their influence on the success of the project. The inclusion of project manager and team members allows the researcher to compare and triangulate different perspectives.

### 4.3 Sampling Frame

Following Eisenhardt (1989), theoretical sampling is used to identify study participants that allows us to test the research model. The sampling frame focused on software development projects that developed novel applications for platforms. Novelty of the application was determined based on whether the application exhibited technological and/or feature novelty.

The sampling approach also focused on identifying software development projects which clearly outlined individual roles. For example, a typical software development team may consist of a project manager, architect(s), user interface designers, developers, and testers. The team may require hierarchical communication pattern, flat communication pattern, or represent a hybrid communication pattern. In addition to defined roles and established communication channels, such software development teams may be subject to organizational requirements related to technology, processes, tools, and approaches. Also, such teams may be subjected to budget and resource constraints. Contrast this to software development by an individual, pair, or small teams where roles are not clearly defined, the team is not subjected to organizational requirements, and budget and resource constraints. Such teams represent entrepreneurs rather than software development teams with entrepreneurial thinking.

The sampling frame focused on identifying software application development projects where the digital platform is known. It is important for this study to identify the digital platform associated with the application because (a) this study is bounded by the context of digital platforms, (b) the research model includes the construct of platform and related subconstructs, (c) core value proposition of the platform can be identified, and (d) novelty of the application can
be clearly defined. Also, the sampling frame focused on software applications such that the application’s platform’s competing platform ecosystems were identifiable. Competing platform ecosystems allow this study to differentiate the core value proposition of the competing platforms. Also, applications available from competing platforms help to identify novelty of the application.

A key point in our sampling frame is the clearly identifiable novelty of the software application. Novelty of the application can be related to its user interface, user experience, data/content, delivery mode, and business-related features. Identifying novelty of the application allows us to focus and distill the key factors considered by the development team to achieve and/or enhance novelty. Given the focus on novelty of the application, our sampling frame is impartial to product-based and service-based applications.

It is important to note that these selection criteria were heuristics used to identify novel software application projects. Our sampling frame does not consider location of the team members, historical revenue or future revenue projects, or business model of the organization because the focus is to identify how a novel software application is developed.

Local software development organizations are considered to identify suitable cases for recruitment. Typically, organizations are contacted via contacts from earlier interactions with the university. Upon access and approval from the management, potential projects in the organization are identified. Each potential project is discussed to identify platform, competition, and novelty of the application. The discussion was limited to upper level management and/or a champion from the project team. The research model and study questions were not revealed to any of the project team members. As a token of appreciation, each participating organization was
guaranteed anonymity in reporting this study’s findings and complete transparency in this study’s protocol.

As potential projects are identified, we also list the individuals which were available for interviews. In some cases, the project was no longer operational and/or key personnel are no longer associated with the organization. Another key aspect for the recruitment of a project was the availability of knowledge pertaining to connected applications. Often applications are released with dependencies across platform’s components and other systems within the client’s systems. Consideration of connected and/or dependencies for the selected project allows us to develop greater understanding of the software development processes. Also, availability/access to project’s documentation, user interface, code, or designs, was considered.

4.4 Recruitment

Two local organizations (abbreviated as AT and TB) are identified as case study locations. Table 3 summarizes the recruited case study locations. One project is identified from each of the case study locations.

AT is a software consulting and contracting firm which specializes in the design, development, testing, and deployment of applications in mobile, web, cloud, and enterprise level solutions. AT consists of industry professionals with different areas of expertise such as application domain, technology, design, and analysis. Typically, the client will engage with AT to identify if AT can provide the right solution to the business problem. After initial approval to develop a technology-based solution, upper-level management will assemble a team. Team members are identified based on the project’s characteristics such as budget and timeline and team member’s expertise and availability.
Table 3. Summary of Case Study Locations

<table>
<thead>
<tr>
<th>Organization</th>
<th>Total Interviews</th>
<th>Informant Roles</th>
<th>Project Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT is a software consulting firm focusing on agile approach to develop, maintain, and deploy software application across different industries.</td>
<td>9</td>
<td>Delivery lead, Team lead, senior developers, UI designer, UI developer, technical architect</td>
<td>The client (a non-profit organization) for this project required a mobile application (iOS-based) which would allow healthcare professionals to stream educational content, videos, support dynamic note-taking, and resume playback.</td>
</tr>
<tr>
<td>TB develops IT solutions to challenges in different domains such as CRM, Healthcare, and Operations</td>
<td>7</td>
<td>Product owner, product manager, practice manager, sales consultant, technical architect, solution architect, functional consultant</td>
<td>The product is a cloud-based Healthcare management application and competes with other offerings on Microsoft Azure platform. TB partners with select customers (hospitals) to develop features which are incorporated into the product – streamline patient care with CRM platform and consolidate patient care.</td>
</tr>
</tbody>
</table>

It is important to note that AT’s approach (as practiced and marketed to clients) to software development is based on Scrum (Schwaber and Sutherland 2016) and DevOps (Bass et al. 2015). These approaches represent the recent wave of agile approaches which focus on continuous deployment and quality. With this stated approach, the development teams follow the agile approach to develop technology-based solutions. Typically, this translates to creation of backlog, sprints, sprint planning, backlog grooming, and use of demos to receive feedback.

Table 4 summarizes the key aspects of the project considered at AT. The client is a non-profit organization. The client delivers educational content to its subscribers. These subscribers are physicians and medical professionals. Currently, the users can access the content via any web browser. To facilitate mobility of the content and better user experience, the client is seeking an
iOS-based application which can be used by its subscribers. The relationship between AT and the client is that of client-provider and is dictated by contract.

Table 4. Summary of the Project – AT Case Study

<table>
<thead>
<tr>
<th>Project Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>Healthcare professionals; geographically distributed; individual use; profile of users known a prior</td>
</tr>
<tr>
<td>Client</td>
<td>Single; Non-profit organization</td>
</tr>
<tr>
<td>Contract</td>
<td>Yes</td>
</tr>
<tr>
<td>Context</td>
<td>Educational content in healthcare domain</td>
</tr>
<tr>
<td>Application</td>
<td>iOS-based application to deliver videos and notes, anywhere, anytime</td>
</tr>
<tr>
<td>Platform</td>
<td>Xamarian; iOS</td>
</tr>
<tr>
<td>Market</td>
<td>Apple AppStore; other applications that offer video streaming capabilities; non-profit’s web-based interface</td>
</tr>
<tr>
<td>Technology</td>
<td>Xamarian; iOS; Microsoft Azure; SQL</td>
</tr>
<tr>
<td>Team’s Location</td>
<td>Mix of office space and geographically distributed</td>
</tr>
<tr>
<td>Approach</td>
<td>Agile (Scrum)</td>
</tr>
</tbody>
</table>

The novelty features for the application were threefold. First, the application allows its users to seamlessly stream content such as text, pictures, and videos on mobile devices. This requires dynamic adjustments to the content. The content is stored on the client’s servers. The native video playback feature from the platform lacked finesse that was expected by the client. The project team developed alternatives to circumvent this problem. Second, client has requested that the users should be able to make and retrieve notes while they are watching videos. Third, client requires resuming the video playback from the last viewed location. This requires constant logging the video watched.

TB is a technology services firm which specializes in business solutions and services. TB offers business solutions across different industries such as finance, operations, customer engagement, and business analytics. A significant percentage of TB’s solutions are cloud-based.
In other words, these solutions are built on cloud platforms such as Microsoft Azure. TB competes with platform providers and collaborates with them across different domains.

### Table 5. Summary of the Project – TB Case Study

<table>
<thead>
<tr>
<th>Project Aspect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>Healthcare professionals; organization-based; individual; unknown a priori</td>
</tr>
<tr>
<td>Client</td>
<td>Multiple; Hospitals</td>
</tr>
<tr>
<td>Contract</td>
<td>No</td>
</tr>
<tr>
<td>Context</td>
<td>Healthcare management</td>
</tr>
<tr>
<td>Application</td>
<td>Browser based application to manage patient offerings</td>
</tr>
<tr>
<td>Platform</td>
<td>Microsoft Azure and Dynamics 365</td>
</tr>
<tr>
<td>Market</td>
<td>Not limited to the platform; users may prefer other competing applications which are developed on other platforms</td>
</tr>
<tr>
<td>Technology</td>
<td>Dynamic 365, Microsoft Azure</td>
</tr>
<tr>
<td>Team’s Location</td>
<td>Geographically distributed</td>
</tr>
</tbody>
</table>

Table 5 summarizes the key aspects of the project identified at TB. The project is developing a cloud-based application for healthcare management. Customers of the application include hospitals and healthcare clinics. Within these customer organizations, users of the application include healthcare professionals such as nurses and clinicians, hospital management, and customer care representatives. The application is developed using Microsoft’s Azure and Dynamics 365 platforms. The application is accessed via a web interface. The application is marketed as a product. Prospective customers are identified from request for proposals (RFP), referrals, or competitive bidding. The team also partners with certain customers to identify new features. Based on the agreement, those features are made available to the customer’s organization while the identified features are incorporated in the application as intellectual property.

The novelty features for the application are threefold. First, the data management in the application is patient-centric. This allows users to view all records for a patient on the
dashboard. Competing applications’ data management is ‘event’ centric—user cannot see all associated events with the patient. Second, the user experience of the application is highly rated. This includes the application’s ease of use and performance. Third, the application can be readily integrated with Microsoft’s productivity suite.

Together, these research sites allow us to evaluate the effectual approach to develop a critical mass of novel applications on software platforms. Also, they differ on team characteristics, nature of application, platform, and technology, on the appropriate development approach for novel applications on software platforms.

4.5 Interview Design and Interview Techniques

The interview process is designed with consideration of two goals: (a) validation of the proposed research model of effectual software application development approach, and (b) identify new constructs and relationships for the study. Interview data will form the basis of elaborative coding analysis (Auerbach and Silverstein 2003). APPENDIX 4 provides the revised interview questionnaire used in the interviews.

Each interview is structured based on the interview questionnaire. However, occasional deviations are permitted to accommodate a contemporary issue. For example, interview questions may be reordered during the interview based on interviewee’s response. Also, interview questions may be dropped in cases where they are not consistent with the role of interviewee. Further, follow up questions (not included in the questionnaire) may be included to seek clarification and/or reconfirmation. Finally, questions exploring interviewee’s role may be included to further understand the process.
Inherent in qualitative research is the ambiguity surrounding certain words and/or statements. For example, team members have different interpretation and view of the application’s features, market, platform, and the overall process. To discern these different interpretation of the phenomenon, our interviews focus on following components:

**Application’s features and components:** As interviewees discuss and describe their role and the development process, interviewees are asked about specific features and components of the application. For example, the video playback feature in AT’s case and connectivity with platforms APIs in TB’s case are some of the components which interviewees can relate their responses.

**Platform’s components:** Another area which facilitates a shared understanding of the process is the platform’s components. The components may be technical (e.g. APIs, tools, programming language), documentation, release cycles.

**Outcomes:** Our research model considers intermediate effects in the effectual process. Multiple sources of confirmation are key to identifying and labeling such intermediate effects.

Table 6 provides a summary of the 16 recorded interviews from two case locations. In case of AT, most of the interviews were conducted at their headquarters whereas a small fraction was via individual online session with capability to share and discuss content. In case of TB, all the 7 interviews were conducted via individual online session.
Table 6. Summary of Interviews Transcribed

<table>
<thead>
<tr>
<th>Transcript (Interview) Number</th>
<th>Case and Role of Interviewee</th>
<th>Duration (minutes)</th>
<th>Number of passages transcribed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AT - Delivery Leader</td>
<td>28</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>AT – Senior Developer</td>
<td>32</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td>AT – Senior Developer</td>
<td>45</td>
<td>62</td>
</tr>
<tr>
<td>4</td>
<td>AT – UI Developer</td>
<td>42</td>
<td>79</td>
</tr>
<tr>
<td>5</td>
<td>AT – Senior Developer</td>
<td>39</td>
<td>70</td>
</tr>
<tr>
<td>6</td>
<td>AT – Delivery Leader</td>
<td>54</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>AT – Delivery Lead (Phase One)</td>
<td>22</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>AT – Technical Architect</td>
<td>36</td>
<td>33</td>
</tr>
<tr>
<td>9</td>
<td>AT – UI Designer</td>
<td>53</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>TB – Product Owner</td>
<td>48</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>TB – Practice Manager</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>TB – Product Manager</td>
<td>65</td>
<td>51</td>
</tr>
<tr>
<td>13</td>
<td>TB – Solutions Consultant</td>
<td>29</td>
<td>40</td>
</tr>
<tr>
<td>14</td>
<td>TB – Technical Architect</td>
<td>46</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>TB – Solution Architect</td>
<td>45</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>TB – Functional Consultant</td>
<td>44</td>
<td>48</td>
</tr>
</tbody>
</table>

4.6 Data Analysis

All interview transcripts were transcribed. Table 6 provides the number of passages in each interview. The data analysis consists of two phases (Miles et al. 2013; Saldaña 2009): (a) open coding, a systematic way to label the data to identify events/actions/interactions and provide conceptual labels, and (b) axial coding, categories and subcategories are related to each other using data.

To analyze the interview transcripts, we require independent coders who have experience in software development projects. This selection criterion is used because (a) interviews include technological jargon, and (b) coding the data requires an understanding of the dynamics in software development projects. We identified two graduate students majoring in Information Systems.
4.6.1 Independent Coders

Before we begin open coding of the 16 interviews, we trained the coders because (a) they do not possess any prior experience in coding qualitative data, and (b) they need to understand and familiarize themselves with the constructs and subconstructs. Operational definitions of constructs and their subconstructs were discussed with the two coders. Any ambiguity in conceptual understanding of these constructs was discussed and clarified. As a training exercise, the coders performed open coding on the pilot interviews conducted earlier. The analysis is performed using NVivo software package. Coders are blind to the research question and model.

For PM1’s interview transcript, the Cohen’s Kappa was 50%. As the Kappa was below the generally acceptable level of 70%, we perform another round of discussion to resolve disagreements and discuss the logic behind agreements in PM1’s interview transcript. Following this discussion, the coders performed another round of coding for both (PM1 and PM2) the pilot interviews. The Cohen’s Kappa was greater than 70% for each of the pilot interviews.

4.6.2 Open Coding

Independent coders coded 16 interviews. Table 7 provides the summary of coding. In addition to labeling the interview transcript, coders were instructed to identify new phenomena that are not part of the list of constructs and subconstructs.
Coders report that it takes about 2-3 hours to code each interview transcript. Coders preferred analyzing the transcripts, first on paper. Following the analysis on paper, the coders performed another round of coding with the NVivo software. Table 8 presents the frequency of each first cycle code. Coders did not identify new constructs and relationships during the coding.
Table 8. Frequency of Construct Coding

<table>
<thead>
<tr>
<th>Construct</th>
<th>First Cycle Code</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Means</td>
<td>Technology</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>Market knowledge</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>Platform knowledge</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>Social Capital</td>
<td>195</td>
</tr>
<tr>
<td>Platform</td>
<td>Technology</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>Market</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>88</td>
</tr>
<tr>
<td>Aspirations</td>
<td>Product-market match</td>
<td>301</td>
</tr>
<tr>
<td></td>
<td>Product-platform match</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td>Exceed Platform Value</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>Novelty</td>
<td>150</td>
</tr>
<tr>
<td>Acceptable Risk</td>
<td>Commit limited resource</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Application recoverable after failure</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Risk Analysis</td>
<td>87</td>
</tr>
<tr>
<td>Logic of Control</td>
<td>Logic of Control</td>
<td>134</td>
</tr>
<tr>
<td>Action</td>
<td>Fixed bugs</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Completed Tasks</td>
<td>15</td>
</tr>
<tr>
<td>Effects</td>
<td>NA</td>
<td>102</td>
</tr>
<tr>
<td>Expanding Cycle of Resources</td>
<td>New technological knowledge</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>New market knowledge</td>
<td>215</td>
</tr>
<tr>
<td></td>
<td>New platform knowledge</td>
<td>94</td>
</tr>
<tr>
<td>Converging Cycle of Constraints</td>
<td>Converging technological constraints</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Converging feature constraints</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>Converging platform constraints</td>
<td>60</td>
</tr>
</tbody>
</table>

### 4.6.3 Axial Coding

The goal of axial coding in this study is twofold: (a) identify relationship between categories and subcategories from the data, (b) validate the revised research model. To accomplish this goal, we
perform within case analysis for the two case studies. The within case study analysis is followed by cross-case analysis to validate the model.

The author of this study and a Professor at USF independently performed within case analysis and then consolidated their findings. The analysis focused on identifying key findings for the case and identifying representative quotes and points for each construct and subconstruct. The analysis also focused on the links identified in the research model. Figure 10 illustrates the process adopted by AT’s team. As a client-vendor relationship, initial discussion pertaining to the application are scheduled with the client. Attendees include key stakeholders from the client and AT’s project leadership. In addition to developing a common consensus on the challenge, the discussion also includes budget and time constraints which may significantly impact the application development.

Figure 10. AT’s Development Process

After the initial discussions, interactive mockups and backlog are developed. These provide a tangible point of reference for identifying agreements, disagreements, and feedback. As stories in the backlog are deemed mature, AT’s development team enter in a sprint of 2-4 weeks. Upon
completion of the sprint, feedback from client’s team is requested. Any incomplete tasks and changes are scheduled in next sprint. Feedback from the development team is logged in the backlog.

During the sprint, AT’s leadership and client’s stakeholders perform backlog grooming. In this discussion-based exercise, the goal is to identify stories for the next sprint. Selection is based on priority, budget, and timeline. New features and requirements identified during the sprint are discussed to decide if and when they should be included in the application. Finally, the client and AT’s team will decide to release the application on platform. In this case, the client is aiming for a specific release date which corresponds to its domain’s conference schedule. Table 9 presents consolidated key points identified in the within case analysis for AT where bold text interviewees are identified as key informants. Table 14 in APPENDIX 6 summarizes AT’s case findings.
Table 9. Key Points in AT Interviews

<table>
<thead>
<tr>
<th>No</th>
<th>Case and Interviewee’s Title</th>
<th>Key Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AT - Delivery Leader</td>
<td>Client relationship, requirements, budget, timeline, align business and development; rapid prototyping to understand business; workshops to identify requirements; discuss backlog items before sprint planning; time and budget influence logic of control; iterative process</td>
</tr>
<tr>
<td>2</td>
<td>AT – Senior Developer</td>
<td>Web API development and maintenance; back-end developer; key aspirations – product platform match and exceed platform’s value; Microsoft Azure used to setup development and test environment which takes away complexity of maintaining these environments; Microsoft core provider; platforms do not provide all functionality;</td>
</tr>
<tr>
<td>3</td>
<td>AT – Senior Developer</td>
<td>Focused on video playback feature; given the time and resources, change aspirations of video player behavior; time sensitive launch for conference; accepted bugs to meet timeline; discuss questions about features with clients; product platform match is a moving target</td>
</tr>
<tr>
<td>4</td>
<td>AT – UI Developer</td>
<td>Development of UI with Xamarian wrapper for iOS; limitations of Xamarian to support video playback; tweak platform components for desired results; AT is recommending rather than decision-making;</td>
</tr>
<tr>
<td>5</td>
<td>AT – Senior Developer</td>
<td>Design, development, and release to Apple’s AppStore; 2 releases, waiting for 3rd release; trial and error; backlog grooming; client decision</td>
</tr>
<tr>
<td>6</td>
<td>AT – Delivery Leader (Final Stage)</td>
<td>Technology-driven recommendations to improve the product; consulting versus contracting; in both situations, AT is order-taking mode</td>
</tr>
<tr>
<td>7</td>
<td>AT – Delivery Leader (Phase One)</td>
<td>Rapid prototyping; feedback from client; get the application to match platform and market, then focus on novelty if budget and time permits; compromise with client on issues that cannot be completely fixed; focus on scope, budget, and timeline;</td>
</tr>
<tr>
<td>8</td>
<td>AT – Technical Architect</td>
<td>Set technical direction; build backlog with ideas and groom backlog to prioritize; clients come up with ideas; technical team brings ideas; ultimate decision by client; subtle enhancements to feature; chaining experiences; new ideas from experience, platform, and technical expertise; testing identifies improvements and ideas for next sprints; application’s usage data identifies focus areas</td>
</tr>
<tr>
<td>9</td>
<td>AT – UI Designer</td>
<td>User interface design for mock ups – 32 screens, Sketch – Mac application to develop; collaborate to identify features; 80-85% features identified prior to development;</td>
</tr>
</tbody>
</table>

Figure 11 illustrates the development process followed in TB’s project team. TB’s team identifies new features from different sources such as conferences, existing customers, users, and
platform’s releases. Together, newer components are debated within the team. Multiple heuristics are used to vet new ideas—technical feasibility, user experience, time required, and intellectual property.

As newer components are identified from a customer, TB’s team may partner with the customer to develop the component. TB’s team then decides if the new component can be incorporated as intellectual property in the product or should be a one-off customization. If the team decides that there exists market for the component, it will incorporate it as an intellectual property in the application.

The team used demos, mock-ups, and proof of concepts to identify features to be developed in the next sprint which are typically time boxed to 2-4 weeks. Sprint is followed by sandbox testing (unit testing in the development environment) and acceptance testing by team’s functional personnel. The newer component is incorporated into the application and deployed and customer’s location. Deployment requires user training, data migration, setup, and connectivity to existing applications in customer’s portfolio of applications. Table 10 presents

Figure 11. TB’s Development Process

As newer components are identified from a customer, TB’s team may partner with the customer to develop the component. TB’s team then decides if the new component can be incorporated as intellectual property in the product or should be a one-off customization. If the team decides that there exists market for the component, it will incorporate it as an intellectual property in the application.

The team used demos, mock-ups, and proof of concepts to identify features to be developed in the next sprint which are typically time boxed to 2-4 weeks. Sprint is followed by sandbox testing (unit testing in the development environment) and acceptance testing by team’s functional personnel. The newer component is incorporated into the application and deployed and customer’s location. Deployment requires user training, data migration, setup, and connectivity to existing applications in customer’s portfolio of applications. Table 10 presents
Table 10. Key Points in TB Interviews

<table>
<thead>
<tr>
<th>No</th>
<th>Case and Interviewee’s Title</th>
<th>Key Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TB – Product Owner</td>
<td>Backlog items from customers, prospective customers, knowledge about the market; prioritize backlog items; 4-week sprints; types of users for modules; Microsoft integration; intellectual property; functional team identifies ideas, discuss with technical architects which provide feedback; remove features that drain performance;</td>
</tr>
<tr>
<td>2</td>
<td>TB – Practice Manager</td>
<td>Implementation; training; Microsoft platform enables scalable and robustness; product platform match is a given as product is built on the platform; 80-90% of functionality is provided by the core product, rest is custom built; Microsoft provide APIs, tools, connectors, and adapters to migrate data</td>
</tr>
<tr>
<td>3</td>
<td>TB – Product Manager</td>
<td>Customers familiar with Microsoft suite; springboard customers; easily find information; data model; ease of use, patient-centric, and open standard interface; all 5 modules on one platform; multiple data points for feature requests; platform release cycle; build on platform release; prototype to debate features;</td>
</tr>
<tr>
<td>4</td>
<td>TB – Solutions Consultant</td>
<td>Registered nurse; pre-sales; nurses, customer service agents, program managers, directors, system administrators; platform is weaved into conversations; license to use application from Microsoft; low visibility in the context</td>
</tr>
<tr>
<td>5</td>
<td>TB – Technical Architect</td>
<td>Offshore development team; in-house development; advanced APIs such as speech recognition; outsource UI design; platform limitations with other platforms at customer side; trial and error – difficult to track; proof of concepts areas in sandbox;</td>
</tr>
<tr>
<td>6</td>
<td>TB – Solution Architect</td>
<td>Small module size; bringing together CRM and healthcare; roadmap; realistic in two weeks’ time; platform enables new tools which are production ready; frequent updates to platform requires refactoring; “we will figure out the platform later”;</td>
</tr>
<tr>
<td>7</td>
<td>TB – Functional Consultant</td>
<td>Agile; partnerships; prioritize items; visual representation; technical input to develop idea;</td>
</tr>
</tbody>
</table>

The cases of AT and TB represent different settings and platforms with commonalities such as uncertainty and novelty of the application. Table 11 presents differences and commonalities across the cases discussed.
<table>
<thead>
<tr>
<th>Construct</th>
<th>AT</th>
<th>TB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nature of Application</strong></td>
<td>Small, mobile application</td>
<td>Large, enterprise-level application</td>
</tr>
<tr>
<td><strong>Means</strong></td>
<td>Team’s knowledge on the mobile application development technology, Apple’s iOS platform, and Client’s knowledge on the users’ requirements</td>
<td>Team’s knowledge on the users’ requirements, springboard customers, Microsoft’s technology</td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>Xamarian and Apple’s iOS provide abstract components which can be used to develop mobile applications</td>
<td>Microsoft’s Azure and Dynamics 365 offer abstract components which is used to develop the application. Well-defined release schedule and documentation</td>
</tr>
<tr>
<td><strong>Aspirations</strong></td>
<td>Product-platform match, product-platform match, product exceeds platform’s core value proposition, and application is novel</td>
<td>Product-platform match, product-platform match, product exceeds platform’s core value proposition, and application is novel</td>
</tr>
<tr>
<td><strong>Acceptable Risk</strong></td>
<td>Team limits amount of time spent in development, meetings, and analysis. Risky components are retained in the backlog.</td>
<td>Team limits amount of time spent in development, meetings, and analysis. Risk is associated with inoperability or performance issues of the application.</td>
</tr>
<tr>
<td><strong>Logic of Control</strong></td>
<td>Application is developed based on known components of the platform.</td>
<td>Application is developed based on known components of the platform.</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Controllable action alternatives are identified.</td>
<td>Controllable action alternatives are identified.</td>
</tr>
<tr>
<td><strong>Effects</strong></td>
<td>Intermediate design artifacts include backlog, proof of concepts, UI design elements, sandbox environments</td>
<td>Intermediate design artifacts include backlog, proof of concepts, UI design elements, sandbox environments, application’s architecture and design</td>
</tr>
<tr>
<td><strong>Expanding Cycle of Resources</strong></td>
<td>Team learns domain specific information to augment technological alternatives</td>
<td>Team learns domain specific information</td>
</tr>
<tr>
<td><strong>Converging Cycle of Constraints</strong></td>
<td>Developed features are dropped; Backlog items are postponed</td>
<td>Developed features are dropped; Backlog items are postponed; Backlog items are revised</td>
</tr>
<tr>
<td><strong>Decision to Deploy</strong></td>
<td>Based on conference/client</td>
<td>Based on client request or team</td>
</tr>
<tr>
<td><strong>Released Application</strong></td>
<td>Provides feedback to next iteration (if requested by the client)</td>
<td>Provides feedback to future version of the application</td>
</tr>
</tbody>
</table>
AT’s application is a small mobile application where a single client provides inputs to develop the application. In case of TB, the application is an enterprise-level application where the team interacts with customers and users to identify components. Differing nature of these applications introduce different forms to constructs and decisions. While the broader means, platform knowledge, and aspirations of the teams are relatable, the focus of identifying these components is influenced by the client (in AT’s case) as against the team’s share (in TB’s case).

Budget plays a crucial role in the decision-making of AT’s application. Although the team may identify multiple action alternatives, the client decides action alternatives based on the budget. In case of TB, budget plays limited role. Other critical decision-making factors include springboard customers and platform’s releases.

AT and TB show similarities in the type of intermediate artifacts developed in the process. For example, both teams use UI designs, proof of concepts, sandbox environments, and backlog to maintain design knowledge. These intermediate artifacts form the basis of identifying new resources and attenuate constraints.

Finally, AT’s decision to deploy the application is influenced by the client’s need to meet a conference deadline. In case of TB, the deployment decision is made by the team. The application is deployed at the customer’s location when requested features are included as part of the base application and customizations. Once deployed, the application is actively used by users which allows the application development team to gather feedback. In case of AT, the next iteration of the application which will include users’ feedback is contingent upon the willingness of the client. In case of TB, users’ feedback is incorporated to revise the next version of the application.
4.7 Study Validation

In this study, we are concerned about four types of validity: construct validity, internal validity, external validity, and reliability.

4.7.1 Construct Validity

Construct validity refers to the validity of the theoretical constructs and their measurement. It is a degree to which measures used in this study can be generalized to the concepts being measured. Two techniques are suggested to establish construct validity: (a) triangulation, and (b) chain of evidence.

Triangulation involves use of multiple sources to confirm the phenomenon of interest. Multiple sources also allow us to confirm the phenomenon from different perspective to gauge the truthfulness of our understanding. In this research, triangulation is achieved by discussing project related ideas, features, and components with different team members. Also, team members have different view of the process. Thus, our data and inferences consider different perspective associated with the platform and software application.

Another technique to establish construct validity is the chain of evidence which allows us to trace backwards, from inferences to the source of findings. To achieve this traceability, this study performed open coding on interview transcripts. Open coding was followed by axial coding. Axial coding identified findings from the coded data to include quotes and summarize key points.

4.7.2 Internal Validity

Internal validity refers to the absence of alternative explanations from those identified in this study. Internal validity is concerned about establishing causal relationships and considers
confounding effects which may be active along with those considered in this study. Three techniques are employed to establish internal validity: (a) use of open interview questions, (b) follow up questions to identify confounding explanations, and (c) multiple informants.

Our interview questionnaire includes open ended questions. These open-ended questions allow the interviewee to identify key information from their perspective. For example, interviewees are asked about their role, novelty of the application, market, and development process. Similarly, other questions seek to identify decision-making heuristics used by the interviewees.

In addition to the open-ended questions, the follow-up questions seek to identify confounding explanations which the interviewees may have missed. Also, interviewees are asked about the efficacy of confounding explanations. Finally, confounding explanations identified by an interviewee are validated with other interviewees. Often, we find that confounding explanations are not shared by other interviewees or identify them as special instances.

4.7.3  External Validity

External validity refers to the generalizability of the results. It is concerned with the applicability of the results in other organization. External validity is established by conducting multiple studies. Our initial preliminary study from Apache Cordova establishes that effectual thinking is present in novel application development projects. Pilot interviews allow us define interview questionnaire. Together, these studies allow us to revise the model. Finally, case studies at two organizations allow us to validate the model.

In addition to the different studies conducted in this research, we also consider different types of organizations. For example, Apache Cordova represents a non-profit open source setting
with disparate contributors. PM1’s organization is a non-profit setting with closed development. PM2’s organization is a Fortune-500 organization. AT is solution consultant organization, whereas TB is a business solutions organization.

Finally, the studies conducted in this research consider different types of platforms and applications. For example, Apache Cordova and AT develop applications for consumer facing platforms whereas PM2 and TB develop applications that are business facing. Variety of these platforms and applications provide support for the generalizability of this study.

4.7.4 Reliability
Reliability refers to the stability of measurements to derive same results under different conditions. This study establishes reliability by providing operational definitions of constructs and subconstructs to coders. Also, use of two independent coders establishes reliability of this study. Table 7 provides Cohen’s Kappa for all the coded interviews.

4.8 Triangulation for Validation
In a software development project, team members have a (slightly) different view on the application’s characteristics and the process adopted by the team. As this study focuses on the software development approach, it is important that we triangulate the software development approach from multiple viewpoints. These differing viewpoints emerge as project’s leadership seek cognitive control over the overall approach whereas team members seek to accomplish the application’s goals. Differences may manifest on decision heuristics and efficacy of decisions in the process. Triangulating these viewpoints allows us to validate the research model.
CHAPTER 5. ANALYSIS OF RESULTS

In this chapter, we look at the evidence to validate the research model and augment our understanding about the constructs, subconstructs, and relationships in the research model. This is followed by discussion of key ideas identified in the effectual software development approach. Specifically, we discuss the tight effectuation design cycles in the software development process and discuss the interplay between prediction and control in effectual software development process. Finally, we conclude with the summary of findings from this study.

Based on the theory of effectuation, we develop a model of effectual software development. To support the model, we performed a preliminary study with qualitative secondary data to support presence of effectual thinking in novel software application development projects. To validate the research model, we designed an interview questionnaire and protocol for a qualitative study with primary qualitative data. To evaluate the efficacy of the interview questionnaire and protocol, we conducted two pilot interviews. The qualitative secondary data and pilot interviews augment the research model. In the following subsections, we discuss our consolidated understanding and evidence in support of the model’s constructs, subconstructs, and relationships.

5.1 Nature of Application

Software applications differ from other software applications in same environment on various aspects such as size, data-intensity, functionality, dependencies on other applications, technology, framework, among others. These differences may manifest due to the application
domain, characteristics of user group, competition, business model, seasonality, if any, among others. The software application’s parent organization exerts a considerable influence on the processes and decisions. Often, the influence is exerted to streamline the application with organization’s broader strategy. Table 12 compares some of the aspects of applications’ nature.

**Table 12. Comparison of Applications’ Nature**

<table>
<thead>
<tr>
<th>Qualitative Secondary Data</th>
<th>Pilot Interviews</th>
<th>Qualitative Primary Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Cordova</td>
<td>PM1’s application</td>
<td>PM2’s application</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td><strong>Data Intensive</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Interoperability with other applications</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Number of Platforms</strong></td>
<td>7</td>
<td>&gt;5</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Non-profit</td>
<td>For profit</td>
</tr>
</tbody>
</table>

We find that larger size of the application induces greater effectual thinking in the development team. Larger applications include greater number of components which serves as a challenge to the development team to develop, maintain, and enhance these components. Different components in larger applications provide different value and novelty to the overall application which allows greater flexibility to manage the aspirations of the team. Smaller applications have limited components which can be developed, maintained, and enhanced. However, smaller applications challenge the development team to introduce value and novelty through these components.

Data intensive applications often require special attention to formats, connectors, and performance capabilities of the application. PM1’s application develops custom components to
handle the data processed by the application. PM2’s application develops custom components to store and process data which are not provided by the platform. In case of some platforms, PM2’s application development develops custom components on top of platform’s standard APIs. TB’s application relies on the platform’s components to include data capabilities with custom hooks for special deployments. We find that in the face of limitations of the platform to provide data handling capabilities, software development teams use effectual process to develop custom components.

Software applications are often required to interface with other systems, before and after deployment. The need to interface with other systems influences the effectual approach followed by software application development team. For example, PM1’s application needs to integrate with client organizations internal systems and external systems from other vendors. These dependencies for data and functionality influence the aspirations, action possibilities, and feedback loops in the effectual process. In case of TB’s application, the development team builds on the applications integration with Microsoft’s productivity suite to exceed platform’s value and introduce novelty to the application.

5.2 Means

In the theory of effectuation, Sarasvathy (2001) identify means of the entrepreneurs as “… traits, tastes, and abilities; the knowledge corridors they are in; and the social networks they are a part of”. For the software development team, means are identified based on the following dimensions.

5.2.1 Technology

Technological means for the development team relate to the technological know-how of the team that has been honed over time. This includes individual-level experiences with other teams and/or organizations and the team’s experiences with prior projects or current project. In
our studies, we find that technological means of the team comprises of industry standards, individual skills, and team’s experiences from prior iterations in the project.

Over the last 2-3 decades, software development standards and techniques have been improved. Recent developments with DevOps has continued this trend to identify techniques to handle ever challenging environments. Often, teams adopt these industry standards to develop software. The technical architect at TB notes,

*We have also used standards on our APIs and portals where we reason, design and development standards that have been around for a long period of time*...

Another major source of technological means for the development team are the tools and environment setups. Specialized tools are adopted for specific tasks in the development cycle. For example, UI designer in AT’s development team will use the Sketch application to design UI components. For a follow up question, he notes:

*Correct, yes, sketch application. But before I get started, I get some basic information from them, like what are the companies brand color, what are their restriction if they have any, so things like that.*

*When I had those at the beginning, then I didn’t design something that nobody would like because it was using their correct color, it was using their branding, it was using the elements that I knew they liked. Or I asked them if I believe to look at their [other application name] application to get an idea of what the [other application name] application design was.*

In addition to the tools used to design specific UI components, AT’s and TB’s team maintain sandbox environments (which may be on a local machine or dedicated server) for trails, proof of concepts, and test new features. These environments enable the team to quickly move through the effectuation cycle and receive feedback for next iteration. Senior developer at AT notes:
Yes, when we prototype it's a sandbox. ... it's a sandbox, probably on our local machine even.

Similarly, the technical architect at TB notes:

... sometimes we have different environments as well. We have a different environment, we have a trial environment, we have a sandbox environment. If something seems pretty straightforward we just build it in a sandbox or that person will just copy the code over from their environment into the sandbox environment and just build it all there. Then we will have a discussion of myself and the person who built it on me that we'll have a discussion as to how we build it, whether it is feasible. Then we will then demo that piece to our product owner Kevin and show it to him and certain marketing people as well.

If we like that sandbox piece then we'll move it to the dev piece and then try building on top of it. A sandbox is mostly a PoC environment. We call it a proof of concept environment. That is where we actually have some of our feature sets and there is a challenge to that as well because it is not a fully baked system. We can do one hand to test in that environment it's strictly a PoC environment.

5.2.2 Market Knowledge

Market knowledge of the team relates to the knowledge and information that the team has acquired about the users, competitors, regulations, and domain specific laws. Often such knowledge base is built over years of experience in the application domain. However, some scenarios such as a consulting environment, the development team may not possess such knowledge. The technical architect at AT notes:

Well, when we get into a project, it's usually a new business vertical that we wouldn't be in this haven't been in this [application’s domain] space for example let's say. Initially it's a collaboration, we learn about the domain, we learn about what the client is trying to do. Then at some point we can bring the
technical expertise and say, "Hey. Did you think about doing this or that"?
Again they goes through the business approval process.

Teams enact different approaches to acquire market knowledge. In some cases, the team may look at competing applications and/or similar application from other domains. The UI designer at AT trials different mobile applications to stay on top of the new UI elements and learn about their usage. He notes:

A lot of personal research. I do a lot of outside project on a daily basis, one of the things I do. I download a new application, for example, every day. Every single day, I'll go to the app store, I download a new application. It doesn't have anything to do with anything I do it can be a game. It can be something about cooking. It can be in any industry any field. I've learned from that, and I've been doing that for 10 years now.

Every single day, I download a new app, and learn from it. That allows me to, one, learn what exist, what new patterns are coming in, see how something works and doesn't work. Just because something is an app doesn't exactly mean it's going to work either, maybe it has a bad user experience. Even from that, I'll learn something or what not to do basically.

A lot of pre-personal knowledge. I've a lot of research, and obviously, working on different application prior to this application.

In addition to these personal endeavors, software development teams also gain market knowledge from conferences and meetings. The product manager at TB notes:

... we looked forward to marketing conferences, user conferences as more the--and also the analyst reports...
In addition to these events, software development teams leverage their existing customers to acquire market knowledge. TB leverages existing customers through partnership, as the product manager notes:

>Matter of fact, one of our springboard customers has been a customer for about four years, but originally they took our [application name] second version of our software and they customized lots of pieces around it. Then they came back to us about 18 months later and said, "We think we've customized too much and we can't really support it anymore. Can you show us where you've taken your tool? Maybe we want to re-implement."

>We showed them our latest tool and essentially they said, "Yes, we want to re-implement with that." There were still a couple gaps that they said, "Hey, would you be interested in a partnership?" Where they provide some market requirements and essentially we provide that back into our base product as opposed to a one-off customization for them.

In addition to such external communication channels to acquire market knowledge, AT’s team includes a solution consultant whose prior work experience is in the role of the application’s user. As a team member, the consultant can provide immediate feedback on the efficacy of the proposed features for the application.

5.2.3 Platform Knowledge

Platform knowledge represent the team’s collective knowledge about the digital platform. The knowledge is acquired over time and includes fine details on the working and integration of the application with the platform. The UI developer in AT’s team notes:

>I think that following standards and protocols in terms of iPhone applications, successful standards that established by Apple and others out there, making sure you’re using controls that people are used to, following practices that people are
used to in terms of phone applications, making sure when you click on something it's user friendly and it becomes something that they understand quickly. That's kind of the thought process that we had.

AT’s experience and knowledge related to publishing applications on Apple’s app store provides important information to develop and deploy the application.

The software development team also acquire knowledge pertaining to different APIs provided by the digital platform. APIs provide an easy-to-use and efficient interface to application development teams on the digital platform. The technical architect at TB notes:

_We are using a lot of latest cutting edge technologies when it comes to AI pieces as well as cognitive APIs..._

_We are using a lot of the latest Microsoft cognitive services APIs. Microsoft has these APIs where you can actually hook your application into and then it does the heavy lifting of actually doing the analysis of all the data it is receiving in the cloud and giving back as code to us, letting us know [the decision]_

The development team learns certain characteristics about the platform and its owners. One of the prominent areas is the release cycle. Technical architect at TB notes:

_They [Microsoft - Platform Owner] have a calendar where they have new releases timeline every quarter. There are clocks or there are if you go on Microsoft's platform, they have a release, a pre-release then a release and then it'll go live of all the features that is coming off and we have-- Sometimes we have access to pre-release version of code which is not available to general public. That kind of stuff happens all the time with us where we start working on pre-release stuff but we don't make it a part of our product until it is for general consumption._

On the contrary, one of the senior developer notes the platform’s release issue:
On mobile, especially with Apple, they’ll just take something away. You've seen them do it with their devices themselves. "Hey, we've taken the headphone jack away," or "I took it away." They don't care. Then the next version, like, "We’re putting it back." Apple doesn't care.

Another area of the team’s knowledge about the digital platform is the use of multiple platforms to develop the application. The technical architect at AT notes:

Like I said, it was not just one platform right here. We were in a way chaining together different experiences like going from desktop to-- and they already had a desktop solution in place. We were trying to have a seamless transition from there to the mobile. For every sprint we need to take all these into account, plus the user's experience in knowing what they already know of the system, right on the desktop. We made it to make that a familiar ground as well.

Similarly, PM2’s application develops connections with multiple platforms in the application:

So there are different APIs. There are [platform owners] APIs and there are [customer] APIs. [Platform owners] APIs is something that we consume from a vendor like [platform owners] etc. [Customer] APIs are what [application name] provides out. And there are also [application name as platform] APIs which our smaller [customers] can actually use to provision [application name]. These are very unique to [application name].

5.2.4 Social Capital

Social capital of the development team includes direct and indirect sources of information which may be internal to the organization or external. A major area of TB’s social capital includes their springboard customers which identify and partner with the development team to introduce novel features to the market. A popular avenue to increase the social capital are conferences which may be domain specific or platform specific.
In case of developers in the development team, the social capital is digital in nature—blogs, messaging boards, community driven question and answer websites, documentation, among others. AT’s development has identified that the native component offered by the platform does not meet the required standard of the application. One of the senior developers was tasked to circumvent this issue with development of custom wrapper. He notes:

*As far as dealing with the technological hurdles, I use the internet like everyone else. Stack Overflow and the Xamarin message boards were the two places that were able to get me the most information on how to get the video player to do what I wanted to do. Unfortunately, the common answer was: don't use that M player, but we didn't have a choice.*

### 5.2.5 Culture

We find some similarities between the teams’ culture in this study. First, teams retain the hierarchical chain of order. In other words, we see defined roles and a hierarchical structure associated with the chain of command. Second, the software development team includes sub teams which focus on specific areas of the application—customer facing, analysts, developers, testers. Hierarchical structures are setup within these sub teams. Third, in addition to formal modes of communication channels, teams have setup informal channels of communication which circumvent the hierarchical structures.

In addition to the team’s culture, the parent organization’s culture has an influence on the software development team. For example, the parent organization may induce a learning culture which is included across different teams. The technical architect at TB notes:

*That is the most exciting part about this process actually. That we every sprint we're learning something new, something exciting and that is what keeps us going. Yes, everyone learns, we learn from a technology perspective. We also*
get a lot of input from the sales people about healthcare vocabulary because we are technologists, we are not clinicians or doctors as such. We also get a lot of input from the sales and marketing as well as we have a lot of nurses working for us.

It only enriches our experience as we keep progressing, making different changes to our product, getting familiarize with different aspects of how the healthcare industry works. As well as on the other side while new technologies that we are using and our tool sets.

Sharing culture within the parent organization also affects the software development team. The solution architect at TB notes:

A lot of the new features, either are already on our road map, our people will talk about what we're going to accomplish this year and how we need to extend the product or what pieces look important. We also listen to our customers. Our customers are probably some of the best people for identifying new features or because they're going to ask once they get used to that now--Now they're going to ask for more stuff. How to extend it and how to make it better. When they are using our tools, what's next.

We're listening to those and we have a group that will, part technical mostly functional that are deciding, these are the things we need to do and we have to be agile enough to change our plans because this is an ever changing situation and what's been good is we've been able to take ideas from one customer, recognize this as more of a--we should be offering this to all our customers so you know that's where we'll work to make it available to everyone. As they get the updates, they get all these new features.

5.3 Platform

This study focuses on software development on digital platform. The digital platform is an external component controlled by its owner(s). The application development team does not
control the platform. In this subsection, we will discuss the components of the platform which are important for the software development team.

### 5.3.1 Technology

The digital platform provides technological components for the software development team such as APIs, tools, and programming environments. Platform’s technology includes components which may not be used by an application development team in focus. These technological components enable the development team to relegate implementation details to the platform. As computation burden is addressed by the platform, the software application can readily use these components to achieve improve application’s performance. In addition to performance improvements, the technological components provided by the platform also incorporate latest technologies which may not be accessible to application development teams. The platform can develop generic components with standard interface and realize economies of scale. For example, the technical architect at TB notes:

*We are using a lot of latest cutting edge technologies when it comes to AI pieces as well as cognitive APIs where we can do speech recognition system as well as doing behavioral, I forget the name for it, but it basically sees how you are, whether you are depressed or not based on certain images that a person sees as well as the intonation in his speech as well and all those kind of machine learning processes that we incorporate in our system to actually come up with a fair indication whether this person needs intervention or he's okay.*

The technological components provided by the platform may be prohibitive in some cases if they limit the application development team’s functionality. In such cases, the development team should develop custom components which circumvent the issue. For example, AT’s senior developer discusses the limitations of iOS’s native video player:
Yes, especially with video on iOS devices specifically. That was something that we had a lot of conversation and they needed a little bit of guidance for. Their expectations were very high and very much based off of existing applications like Netflix and the YouTube applications. What they maybe didn’t understand is that Netflix and YouTube and Hulu and what not went around the native iOS video player and made their own, because at the time -- and probably still, the native iOS video player is pretty terrible.

They really want you to use HTML5 video players but within the time and the scope of the [client name] project; that wasn’t really an option. Using the native M player and iOS was what we had to deal with. You don’t get all those really fancy features that Netflix and the YouTube app give you. Working with them on that was -- they were very understanding once they got the gist of the fact that Netflix and YouTube’s business is making video players; your business is just presenting video content, so maybe scale back your expectations for the first iteration of the project at least.

With such limitations, the development team needs to develop custom components to circumvent such limitations. However, such custom development requires maintenance over time. AT’s senior developer on circumventing this issue:

I can tell you from the video player perspective, we did beat that thing into submission to get it to do things that it’s not normally able to do. That was an interesting challenge given the time and the platform that we had, it was by no means excessive. I wasn’t working a hundred hours or anything, I was able to do a normal 40 hour work week, but Xamarin is a wrapper around the native iOS components so there’s two layers of abstraction. Also, I'll say that Xamarin is an open source project and the video player is not something people focus on. Already, we were caught up against the wall but we’re able to really get that thing to do what they needed it to do within the amount of time that they gave us.
5.3.2 Market

Digital platforms provide a market where users and application developers can interact to participate in value adding transactions. Applications are typically categorized under labels to facilitate matches. Users can browse these applications to identify applications which match their requirements.

For a development team, the market provides competition and learning opportunities. Other applications in the same domain serve as competitors. As the platform does not differentiate among application developers, the competing applications have the same access rights and performance capabilities as that of the team’s application. Challenges pertaining to the platform, faced by the development team are also faced by its competitors. This includes changes to platform’s interfaces, licensing costs, and component limitations.

5.3.3 Core Value Proposition

The digital platform offers components to enable generativity (Zittrain 2006). Software development application teams can use these components to develop newer components or use them as is. For example, AT used existing components to alter the video playback feature discussed earlier. Similarly, the platform allows adaptation of components to the application’s domain. TB’s product owner notes:

Because our solution is based on a CRM platform, we have a capability to record an interaction with a patient. A phone call can be logged. It’s integrated with email. In that way, it helps streamline, and also coordinate the patient interactions for monitoring chronic care conditions.

Digital platforms also enable the software application development team to incorporate other applications’ functionality. For example, TB’s product owner notes:
One is that because we’re on the Microsoft platform we are integrated with all of the Microsoft dynamic’s solutions and office solutions. As I mentioned, we can integrate with email. We can integrate with Power BI. We can leverage the – For example, they have a new relationship’s analytics, so that’s machine learning in the background. We can leverage all of those things, and ongoing enhancements that they make the Dynamics 365. And leverage all of that into [application name].

Interestingly, the platform components are generic to the extent that they do not differentiate between categories of products. For example, the product manager at TB notes:

The CRM really provides a set of standard sales service and marketing capabilities that are not specific to any one industry. We have leveraged that and built on top of it to make a healthcare provider specific version called [application module name]. Part of that solution is really in Dynamics 365. It is essentially a self contained configuration of Dynamics 365. But the way that's built we can essentially package that up into what Microsoft refers to as a managed solution and then that becomes a read only component that we can deliver to our customers.

The digital platform also provides value to the application development team by enable rapid scaling and flexibility in production environments. For example, practice manager at TB notes:

...if you’re looking to really scale up or scale out to other divisions within your organization while maintaining a single platform that integrates with other Microsoft platform and offering seamlessly right because it's all Microsoft products that we're working with. I think that it just makes us much more flexible and scalable and much more robust operates

Similarly, the practice manager notes flexibility offered by the platform:
But because we are on the Microsoft platform, we are also flexible to add or modify. We take the product and we configure it to meet the various needs. If there is something that the product doesn’t do there’s often times something that we can do with the platform.

5.3.4 Maturity

Maturity of the digital platform refers to the predictability of the platform with respect to release cycles, support for components, and generativity offered by its components. TB’s team notes that the digital platform releases twice every year. These releases are preceded by pre-releases and demos. However, the solution architect also notes lack of maturity by the digital platform in earlier releases:

One of our problems with Azure is they have five names for the same thing. Yes, we do. We are listening a lot more to what's coming from Azure. In the past, a lot of the tools probably a year ago they were --it almost seems like they were research products, right. They were--I hear something and if you could make it work could be great. Now they're becoming production ready.

One of the reason, as discussed earlier, is the type of consumers for the platform. AT’s team note the fast-moving nature of Apple’s platform and abrupt removal and introduction of components. These changes are required given the consumer facing environment which introduces new hardware and software at rapid pace. In case of TB, the platform interacts with businesses where software components change at rapid pace but can be incorporated with fewer changes.

5.3.5 Complexity

Complexity of the platform refers to the ease with which software development teams can use individual components of the platform. High complexity implies difficulty in integrating the application with the platform due to lack of decomposability. On the other hand, lower
complexity implies that the application development team can integrate seamlessly with the platform by identifying specific components. In our studies, all the platform’s demonstrate low complexity. Tiwana (2013) discuss the importance of modularity to address complexity of the platform.

5.4 Aspirations

We identify four broader aspirations for the software development team.

5.4.1 Product-Market Match

The aspiration of product-market match has been considered in the literature in software development. In a platform context, the software development team is required to match the market with a consideration of platform’s constraints. The software development team may identify certain features required to match the match via research or client may detail (if any). For example, in AT’s application, the client provided some upfront information about the key requirements for the application. The team also augments this with their understanding, as noted by UI designer:

...lot of research up front on my client's business when we start their market, who they are, what they are trying to achieve, who their competition is, how the users are currently responding to their business, and how we want them to respond in the future. The goal is to think for the user.

In case of TB, the team understands market needs during sales, conferences, and deployments. For example, the practice manager notes:

Some of it is hearing things in the marketplaces as far as the shift but I think a lot of the feedback that I see is from customer based or partners that we have. We definitely try and listen to the customers and then their demand. Sometimes
it's the pre-sales engagement where they're asking for something or they're looking for something unique and we put it more than once or twice. It's something that keeps coming up then that really gets taken along more seriously as to should we just incorporate into the product.

While the software application team strives to develop an inclusive product, some niche requirements may not be matched by the application. In such cases, the development team may include a custom component. However, if the demand for such component is widespread, it may be incorporated in the base application. The practice manager notes:

“There's always something unique it's just how extreme. Let me clarify, usually when we do [application name] implementations, the products gives us 80-90% there. Then a customer will always have something unique around maybe they need to add a specific set of field to a patient record or a facility record, or maybe I need to have a custom type of activity that happens within my workflow processes. But the functional aspects of the application typically gives us 80-90% there.

5.4.2 Product-Platform Match

In a platform context, product-platform match is the starting point. In other words, the application can only function and serve its users if it is compatible with the platform. As the technical architect at TB notes:

Our decision is always driven by whether it will work on this Microsoft platform. Any new feature as such. If the sales team comes up with certain requirements that it is totally out there, out of left field like this application to launch rockets in the sky, it's not going to happen. It is always driven by how feasible this solution is going to be on the platform.
There are instances where the application may not match the platform. That is, the platform may not support certain features which the team would like to implement in the application. In such cases, the team may develop custom components (e.g. video player by AT’s team) or accept it as a limitation, as technical architect at TB notes:

> There are instances that in certain cases Microsoft's platform blocks out certain windows for some reason and yes, you have to fight those battles and then go to the customer and say that yes, this platform does not support this device.

The application development team often balance the limitations of the platform, as product manager at TB notes:

> That's a little bit of the trade-off to say, "Well, do we need what's technically possible from the platform, or what's visually possible from a design agency?" We've tended to err on the side of, "Let's show the users what's possible, and then challenge our technical teams to fit that into the platform." It's driven more by the end-users than our architect that says, "Well, that's to hard," or, "that can't be done," or, "that won't work in the current release."

In addition to the application development team’s motivation and need to match product-platform, certain platforms may mandate application’s review before they can be deployed and made available to users on the platform. For example, AT’s application is subjected to a formal review process by Apple. During the review process, Apple may require AT’s team to change certain components which may not adhere to its guidelines. AT’s team is mindful of this process and accounts for it during their planning. Conversely, other platforms may not require formal reviews. For example, TB’s application is built on Microsoft’s platform where formal process of review does not exist. The application is built and deployed at client’s location.
5.4.3 Exceed Platform’s Core Value

The application development team exceed platform’s core value proposition by incorporating domain specific actions. For example, product owner at TB notes:

*We are trying to exceed the Dynamics 365 value by making it tailored to healthcare because there are not a lot of other products that do that. I would say some are for -- Not as much by the nature of being integrated with the Microsoft platform, and all of Microsoft products, that makes us pretty novel.*

AT exceeds platform’s core value proposition of the video player by developing custom wrappers to enhance its functionality. Senior developer notes:

*As far as exceeding the core value of the platform, I can tell you from the video player perspective, we did beat that thing into submission to get it to do things that it's not normally able to do.*

In case of AT, the development team is required to facilitate discussions with the client to consider the value proposition of the platform and how the application can exceed it incrementally. UI designer notes:

*It's being able to push that and change it little by little to show them the value that you're adding, and why you're changing it because that's one of the thing, if you build something, by the time you're done building it, it's already outdated. Being able to teach them that and show them that how you have to add value to it, and how to update a design on what new API's being offered by the operating system to be able to do more things.*

5.4.4 Novelty

We identify the application’s content as one of the major source of novelty for AT’s and TB’s applications. Content refers to the data, information, and knowledge, delivered by the
application. While the content of the application may precede the application, the key differentiation offered by the software development team is to design, develop, deploy, and realize the potential of the content.

In case of TB, the application’s data model (patient-centric rather than event-centric) allowed it to develop novel features which are in demand from customers. TB’s team also identify the scalability, flexibility, and robustness, offered by the application as a novel feature. The Microsoft platform allows the application to integrate other Microsoft productivity tools with the application. Finally, TB’s application can use other platform components such as Power BI, speech recognition, and analytics, to develop a novel application. Solution consultant at TB notes:

*I believe is unique about our solution is that smarter approach to providing one place for everyone to manage and coordinate the care of that patient no matter how many conditions they have, no matter how many health care providers they are seeing, it's one care plan. Then we resolve duplicate goals and things like that so there is some intelligence behind the scenes that says, “Hey, this is the exact same goal for hypertension as diabetes. Let's not obviously show it twice.”*

5.5 **Action**

We identified two subconstructs for team’s actions: fix bugs and completed tasks. Bugs introduced in the development process are addressed by the development team. These include issues related to UI elements, workflows, and application performance. Completed tasks include feature requests which are incorporated in the application. Typically, issues are tracked using project management tools such as JIRA. Issues are assigned to specific team members who analyze, develop solution, and implement the solution in the application.
We find that software development teams use discussions to identify and consolidate action possibilities for the team. These may be in the form of sprint planning, backlog grooming, scheduled sessions, or impromptu sessions. The goal of these discussion sessions is to (a) identify action possibilities for the team, and (b) selection of appropriate actions to pursue.

Identification of action possibilities is based on the team’s means, platform, and aspirations of the team, identified at any given time. The team’s means identify the set of technological possibilities for the application, given its knowledge about the platform, market, and domain. The platform may identify certain action possibilities by changes to its components or introduction of new APIs. Finally, aspirations of the team identify features and actions which are desired by the team. Although the action possibilities are perusable, the team decides to act on actions based on two heuristics: acceptable risk and logic of control.

5.6 Acceptable Risk

The development team identifies actionable items by committing limited resources. Resources may be in the form of budget, time, or functionality. The technical architect at TB notes:

*They come up with a wish list and then we actually then deliberate over what is there in the list and some of them are pie-in-the-sky requirements which is not technically feasible. Then we actually come up with what is feasible, what's the timeline that will take all these things to be done. There is a back and forth going on. Then based on that budget constraints, resource constraints, we come off like a proud list of what we can achieve and who does what and come up with a plan.*

Similarly, the technical architect discusses the budget limitations which identify actions:
The process happens really during the sprint planning. Then let's say if it is a small tweak to the feature like, "Hey we can do this little improvement, it's not going to cost us a lot." then we just add that to that feature. Now if it is a larger enhancement and the product owner is not sure if they want to spend the money on that right then, or are at the time like let's say, "Hey that's a cool feature but we want to save it for not the MVP, not the first release, but the later release." In that case what they do is they capture the idea and then put it as another feature on the backlog, then move on. Now this was suggested, it came about and we captured it, they have it in there and then it's saved for later and then the current sprint continues as planned.

The software application development team is also mindful of integrating new components and features in the application and its impact on the current version of the application. For example, the technical architect at TB notes the severity of maintaining the sanctity of current version:

We are pretty rigorous about our testing about regression testing a solution with all this new feature. If it starts bombing out or if it has a negative impact on performance then we actually roll back those changes. At the end of the day it has to work with seamlessly work with the product. If it compromises our product and then it is out.

Similarly, AT’s senior developer notes:

To me, the three most important things are from a developer standpoint and performance. Does it do what it's asked to do and does it do it well? Is it maintainable?

We're consultants. I've been a consultant or a contractor for a lot of companies, so ultimately, I write it and then we give the code to the customer. I don't think we did that in the case of [client name], but it has to be long-term maintainable. I've got systems that are still running that I wrote 12 and 15 years ago that
people still have to work on. When I take into account the code that I'm going to write: is it maintainable? Then finally, is it extensible? Can they build on top of this?

Software development teams also consider the risk associated with action possibilities. Primarily, these are associated with the application’s performance and changes to the platform’s components. The product owner at TB notes:

...there's always that risk of breaking something. I always feel like the risk is usually something technical. We don't want to mock around too much with the standard platform because then things don't work as well. We also have had to forego some enhancement because they really drain performance, so that’s a big one. Yes, I would say we certainly have to weigh risk. It's usually around the architecture or performance.

The functional consultant at TB notes the risk of changes to platform’s components:

...another risk or concern is if we build everything now before the new update rolls around, what if it’s not compatible? I feel like that’s always a common risk. What if our code doesn’t work there? I know right now we’re encountering this with the new Microsoft CRM updates.

I know we’re working on trying to tie things down and make sure everything is good and make sure everything is compatible in the next release, but luckily we do hear a lot about what’s coming in the future releases so we can try to prepare as much as possible, but it all really depends how quickly do we need to build this? If this can wait out until the next release to build it, then I think that’s a team design decision that needs to be made.
5.7 Logic of Control

Software development teams follow action possibilities which can be controlled given their means, platform, and aspirations. The solution architect notes that the team identifies the feature which will be incorporated in the next release of the application:

Well we decide what's going to a release. We will pick a number of things. We'll also collecting the next version, right? As you're working on one you're listening and collecting and thinking of the next extension. We do do that. How formalized that is based on which one of the subsystems that we're working on. Some are bigger than other. Healthcare has a lot of standards for compatibility around data exchange. We're very cognizant of those.

The application development team also consider the platform’s components which can be controlled/modified to suit the application’s needs. A delivery leader at AT notes:

I believe there was a few items that we had thought Apple would have released during the development cycle but they did not, that feature got taken out. I don't remember what that was. Now I go with no more and she could explain that, but yes absolutely it's a constant shuffle. It's a constant prioritization based on business value, based on development time and based on the availability of the technology.

5.8 Effects

An effect is the intermediate artifact developed from actions of the team. Our study highlights different intermediate artifacts developed by the application team. First, software application teams use mockups to illustrate, identify, and provide feedback, in the development process. Mockups are in the form of UI elements and screenshots that can be evaluated by team members.
Second, backlog represents a central repository of the team to log, update, and implement, features identified and updated in the development process. Backlogs describe user stories which consists of details pertaining to the development and efficacy of the requested feature. User stories may be identified a priori and during development, as the team identifies avenues of improvements. Delivery leader at AT notes:

*The development cycle, the way that our development cycle works is we work in two weeks sprints. We develop software every two weeks, we demo the product to the client for that sprint whenever we accomplish that sprint. Prior going into that sprint, we have the backlog grooming session. You groom all the stories and requirements in a manner so that they're available for the next sprint.*

*During the start of the next sprint, we have a planning session. We look at all the backlog requirements that have been approved by both the business and buyer technical architect that they're approved and they're ready and then we bring them into the sprint. That iteration happens every two weeks.*

Third, development team creates documentation in the form of technical and design documents. These documents are a significant source of reference for the team and a repository update the application as new platform components are available. Documentation also helps to forge a common understanding of application’s components and enable maintainability of the application.

Fourth, software application development teams perform proof of concept exercises. These exercises may include setup of specialized environment to test the feasibility of new features. Upon confirmation, these act as intermediate artifacts which enable the team to decide if they would pursue the feature, develop it, and incorporate it in the application. Any failures in the proof of concept environment also serve as intermediate artifacts, as they enable the team to
incorporate result. Related to these exercises are estimates of effort and cost for the development of components.

5.9 Expanding Cycle of Resources

Intermediate artifacts enable the software application development team to expand its resources. Technologically, the team can identify new components and assimilate changes to components. The technical architect at TB notes:

Whenever we come across something that we see from the technical side that could add value to the product we definitely bring it up. Then it is discussed within the team and then it becomes a part of our future wish list but how it gets formalized also then depends on resources.

Intermediate artifacts also help the development team to identify future technological components. Senior developer at AT notes:

Capturing all those events in iOS, and in iOS, the user delegate system was challenging. I definitely learned a lot about how the iOS delegates work and how to capture all this kind of stuff. Then adding again the layer of abstraction of Xamarin, not writing in native Swift or Objective-C code definitely made it a little bit more challenging.

The development team updates the backlog with new features and update existing stories based on improved understanding through intermediate artifacts such as mockups and documentations. The delivery leader at AT notes:

I believe if I remember correctly, they had a prototype coming into this of what they thought their vision would be. Of course, that evolved and became once you got it into an interactive mockup. It became more real so that changed but there were some ideas but nowhere near ideas that they go to market with.
Intermediate artifacts also help to improve the teams understanding of resources and limitations of the digital platform. The product manager at TB notes:

*We've been investing quite heavily in their releases. They follow essentially a spring and a fall release, so two releases per year. We follow those releases building on top of their functionality. I think really what's most beneficial to us is more of the access rights, the privileges, the controls that they provide, the ability to deploy on a tablet with one code base. So, I think it's really more the tactical underpinnings that they provide.*

### 5.10 Converging Cycle of Constraints

Intermediate artifacts identify technological components which can be used in the application and those which cannot be used. Typically, the components and features which are not supported by the platform force the team to identify other means to incorporate those features, abandon them, or revisit them later. The product owner at TB notes:

*...the functional team comes up with the dream, "How do we want it work ideally." Then we meet with the architects, and say, "Okay, this is what we want. Is that possible and if so, how?" Then usually, they realize, and then say, "Well, we can't do it that way, but here's what we can do, and how about let's do it this way?" Then, usually, they'll write up them notes, and then they'll send that to [product manager] or I to approve, and say, "Is this okay? This is how we decided we want to do it."*

Intermediate artifact also enables the team to identify feature they may add to the application, update features, or remove. Such decisions on existing and new features may be triggered based on the team’s evaluation of intermediate effect. UI developer at AT notes:

*In actual world they will ask for a feature and they'll look at it and then like, "Okay. I like this, I don't like this, let's make this change, let's do this differently,*
I liked it before, but now that I see it in actual applications, it's not as good as I thought," or something they thought they didn't like, they like better when they actually see it and reiterate around on it again and make changes they request. Sometimes we as consultants we would recommend things especially with standard practices in terms of how user expects the application, the function comparably to other apps out there and stuff like that.

Converging features may introduce further changes in the application as the team evaluates the application as a whole. The product owner at TB notes:

At any point, when we make a change to one thing, it might mean we need to make another change to make it flow. So oftentimes in testing or in creating the outline of changes that need to be made for current users stories and requirements, anybody on the team finds something that should also be enhanced.

Evaluation of intermediate artifact also enables the team to identify platform’s components which may be used to include newer feature or identify deprecated platform components which may no longer be supported. UI developer at AT notes:

That was a challenge because sometimes at the time it was a year or so ago if I recall maybe a little more Xamarin didn't support as many iOS like everything that iOS supported out of the box. There was some things that didn't support. It's for the most things but a lot of the times I found and just by when I was working going along, I need this type of control with this functionality and I would see what was available to me right out of the box and then I would either find it or I wouldn't and then I would hit the documentation online or I would reach out to others that here that have worked on it and see if they have any recommendations.
5.11 Decision to Release to Platform

Number of factors play a key role in the decision to release and/or deploy the application. Broadly, these factors can be aggregated into three categories: market-driven, platform-driven, and team-driven. First, market-driven factors which may decide release of the application include conference, trade shows, regulation, and request for proposal deadlines. These factors cannot be controlled by the team. Second, platform-driven factors include release of application after incorporating changes to the platform. As platforms update their components, the application development team may be forced to release an update to retain the application’s compatibility. Finally, team-driven factors include budget constraints, team culture, and organizational factors. Teams may adopt a policy where they release an updated version of their application every week or month. Similarly, organizational policies may dictate the periods when the application can be released. Application development team collectively consider these factors, although they may appropriate differential weights to these factors.

5.12 Released Artifact

The application released to the platform or deployed at customer’s location is actively used by its users. As users use the application, the application development team may receive feedback on components from different channels. Such channels may be word-of-mouth, formal reports, reviews, and competitor analysis reports.

In addition to these channels, the application development team may identify application’s components which need alterations. For example, AT and TB use application analytics to identify components which are regularly used by its users. Analysis of usage patterns often serves as an evaluation of their understanding of users’ processes. Analysis of usage data also includes errors and issues pertaining to performance of the application. Together, such
feedback is incorporated in the effectual software application development process to identify new components, update components, and drop components.

5.13 Summary

In this chapter, we have discussed evidence for each construct and subconstruct in the revised research model of effectual software development approach. In the following chapter, we discuss four research implications of this research. The research implications identify broader contributions of this research to the information systems research.
CHAPTER 6. RESEARCH IMPLICATIONS

In this chapter, we discuss four research implications of this dissertation. First, we discuss novelty of the software application and how effectual thinking can contribute towards this goal. Second, we discuss the tight effectual design cycles and their importance in the effectual approach to software development. Third, we discuss the balance between prediction and control achieved by the development team. Also, we discuss how this balance evolves across phases. Finally, we discuss artifacts which support the planning and execution and monitoring phases of the effectual approach.

6.1 Novelty of the Application

Traditional software development projects focused on the project’s performance which is typically identified as budget, timeliness, and quality. The environment in which the application is to be deployed is assumed stable. With increasing focus on fast-moving markets, software development projects focused on product-market match. The challenge is to develop a relevant application in face of market and technological uncertainty. Recent focus on digital platforms has introduced novelty of the application as a key success criterion in addition to traditional criteria of success.

We defined novelty of the application as its features and extensions which are not provided by the digital platform and competing applications. In a digital platform ecosystem, novelty of the application assumes importance as it allows the application to distinguish itself from other competing applications which have similar access to platform’s connections. Further,
novelty of the application evolves over time as platform’s core offerings and competing applications may incorporate novel features of the application. In this research, we conceptualize the software development team as entrepreneurs.

Other approaches to software development have limited applicability to introduce novelty to the application in uncertain, risky, and resource-constrained environment such as digital platforms. First, plan-driven approaches identify the software specification and discourage any changes to it in latter stages. Thus, novel features are required to be identified a priori. Second, ad-hoc approach relies on individual contributors to identify novel features and incorporate them in the application. The ad-hoc approach does not identify a process which can guide the software development team to identify and assimilate novel features in the application. Finally, the controlled-flexible approach focuses on defining scope boundaries and facilitating ongoing feedback. Scope boundaries provide the control mechanisms whereas ongoing feedback allows course corrections during the process. Iterations allow the development team to manage the development process. As discussed earlier, the primary motivation of this approach is to position the application/product in an uncertain and fast-moving environment. Novel features identified by the development team during iterations may be incorporated in next iteration.

The effectual approach to software development advocates a different approach with focus on existing means and aspirations of the software development team. Action alternatives with acceptable risk and controllable profile identify intermediate effects. Intermediate effects are tangible design artifacts which expand the resources and attenuate aspirations of the team. This effectual cycle emphasizes the team’s resources to identify, incorporate, and expand novel features in the application. It also supports serendipitous identification and evaluation of ideas for the application (Austin et al. 2012). Fast effectual cycles provide a process through which
ideas are identified, evaluated, designed in the application, developed, tested, integrated, and deployed with the application.

We find that the key to introduce novelty in the application is the feedback loop between aspirations of the team and design artifacts. Aspirations are abstract representations of the team’s end goals. Different aspects of team’s aspirations (product-platform match, product-market match, novelty, and exceeding platform’s core value) identify application characteristics and their relative importance to the context. Aspirations also identify action alternatives which will accomplish desired aspirations of the application. Further, aspirations serve as evaluation tools for the team. Intermediate design artifacts identify aspects of aspirations which are attainable and those aspirations which are not attainable, given the resources at hand. Evaluation of intermediate design artifacts allow the application development team to fail early at lower cost because the team is evaluating the intermediate design artifacts with its aspirations.

Novelty of the application may be identified a priori and/or over time as the application is developed and deployed. In case of AT, the content delivered by the application is considered as its novelty. However, other novel aspects of the application are identified during the development process such as ability to make active notes and video scrolling. Similarly, in TB’s case, a priori identified data model introduced novel features and capabilities in the application. However, other novel features such as analytics capabilities, artificial intelligence, and reporting, were identified during the application’s development. Thus, the effectual cycles are key to identify, evaluate, and incorporate novel features in the application.

It is important to note that our studies identify three broader categories to identify novel components of the application. First, by virtue of their role, team members in the visionary quadrant identify significant proportion of novel features in the application. Typically, these
roles are occupied by individuals who have experience in the market, application, and interact with customers on regular basis. Second, novel features for the application are identified by developing partnership with representative customers. In case of TB, the application development team partnered with springboard customers to identify novel features for the segment. In case of AT, the client served as the representative customer. Customers who are willing to enter in partnerships not only identify novel features but provide feedback which can evaluate the efficacy of the application. Finally, the technical team may also identify novel features for the application. As the platform releases new components, the technical team may recommend changes to existing application components.

6.2 Effectuation Design Cycle

The revised model illustrates the effectual design cycle followed by the software development team. The team considers what the team knows, what it wants, and the generativity offered by the platform. Based on these means, aspirations, and platform’s state, the software development team identifies action possibilities. The software development team identifies action possibilities which embody the logic of control and acceptable risk profile. Action results in intermediate design artifacts which provide feedback to identify new resources and attenuate aspirations. Intermediate design artifacts also serve as evaluation of team’s aspirations.

We see that the software development team enact tight effectual design cycles. These cycles are labeled tight because they may be enacted through compact processes which may include few personnel and/or decisions. For example, an effectual cycle may include evaluation of proposed feature with proof of concept. The time taken from idea inception to evaluation via proof of concepts may be a few hours. Sprint planning and backlog grooming are other example
of tight effectual cycles. The effectual cycles produce a tangible outcome which can be evaluated by the team.

Effectual thinking is embedded within these tight cycles. This allows the software application development team to grow its knowledge and shape its environment. This logic is different from other plan-driven and ad-hoc approaches which focus on positioning or purely adaptive the application to maintain relevance and profitability.

The effectual cycles enacted by the software development team are cycles which improve the design knowledge of the application development team. The cycle corroborates with the design cycle discussed by Hevner (2007) where design artifacts are developed, evaluated, and feedback loops inform the next cycle. Simon (1996) relates to this cycle where alternatives are identified and evaluated until a match with requirements is achieved. For software application development on platforms, changing requirements require tight effectual cycles which enable the team to identify appropriate alternatives. Effectual thinking informs the software development team to identify alternatives and decision processes to evaluate.

The effectuation design cycle identified in this research corroborates with recent work in research processes. Mullarkey and Hevner (2018) elaborate (eADR) on the Action Design Science (ADR) method (Sein et al. 2011) to identify multiple iterations of Problem Formulation/Action Planning (P), Artefact Creation (A), Evaluation (E), Reflection (R), and Formalization of Learning (L). These iterations are four stages of diagnosis, design, implementation, and evolution. The effectual approach discussed in this research aligns with these ideas. Specifically, the effectual approach identifies aspirations for the application to be developed. Based on identified action possibilities, intermediate artifacts are developed and
evaluated for their match with the team’s aspirations. The team reflects on intermediate effects to expand its resources and attenuate its aspirations.

Mullarkey and Hevner (2018) also note that eADR allows multiple points of entry. Similarly, the effectual approach allows multiple points of entry to the software development process. Specifically, new aspirations and means which are not identified in the effectual process can be incorporated in the process. For example, in AT’s case, mandated release date of the application was identified as a requirement for the software development team. Similarly, development of alternate video player during the process was also incorporated.

This research contributes to the discourse on design cycles. First, effectual approach highlights the importance of existing means and aspirations of the team to develop intermediate design artifacts. Second, effectual approach considers broader design artifacts to include proof of concepts, working code, prototypes, design interfaces, among others. Third, prior design cycles include conceptualization, development and evaluation of artifact as key phases. Effectual design cycles enumerate these phases at greater speed. Finally, effectual design cycle is tight. Each iteration focuses on evaluating a specific aspect of team’s aspirations. While an iteration may help the team to evaluate a specific aspect of team’s aspirations, evaluation of intermediate design artifact increases the team’s knowledge to better position future design cycles.

6.3 Balancing Prediction and Control

In chapter 2, we discussed and analyzed the theoretical underpinning of software development approaches using the framework of control and prediction. Specifically, the plan-driven approach to software development follows a planning strategy to position the application for relevance and incorporate small but large controls. The ad-hoc approach emphasizes rapid adaptation to maintain relevance. A controlled-flexible approach balances planning and adaptation. We
concluded that an effectual approach can control and shape the environment for the application to remain relevant by incorporating many but smaller controls. In this subsection, we revisit these notions, considering the case studies.

Research in the management of software development projects has considered pros and cons of different approaches and their suitability to specific environments. Across these studies, the assumption is that all team members follow the approach identified by the project manager. However, we find that team members incorporate adherence to different approaches. Further, based on the application’s current state of development, team members alter their adherence to different approaches. This suggests that although the team may adhere to identified approach by the project manager, individual team members adapt within the framework defined by the project manager.

Figure 12 superimposes AT’s team members’ roles on the framework of control and prediction. A role is assigned to a quadrant if the team member illustrates the underlying logic of the quadrant’s strategy. Based on the UI design provided by the designer, the UI developer a priori identified the tools and components that he will use to accomplish the application’s UI. Over the course of application development, these components did not alter. Similarly, the delivery lead identified specific aspects which were identified a priori such as budget, timeline, and key features of the application. However, we also note the delivery leader in the transformative quadrant as he facilitates the sprint reviews and backlog grooming.
The visionary quadrant strategy predicts future environment and control it to shape it. The product owner (client liaison) provided thought leadership for the application and identified a vision for the team. To realize this vision, the UI designer developed designs based on components which were identified based on the team’s means. Similarly, the technical architect identified technology (programming language, tools, and framework) to realize the vision. The technical architect, also identified in the transformative quadrant, uses control-based approaches (prioritizing, analytics, feedback) to identify features which may no longer be used in the application. The senior developer is identified in the transformative quadrant as the team member developed the complicated video streaming custom component to circumvent the platform’s limitations.
Figure 13 superimposes TB’s team members’ roles on the framework of control and prediction. The practice manager implements the application at customer’s location. Implementation involves planning data migration, migration tools, and connectors. Product manager and owner identify certain features and components which may be added to the application. These components are identified based on their understanding of the market, interactions with customers, and analysts reports. In the visionary quadrant, the product manager and owner incorporate the inputs received from external avenues and envision the possibilities for the application. The solution architect identifies similar avenues for the application to expand given the new architectural possibilities from changes to the platform and application.

In the transformative quadrant, the technical architect identifies alternatives which can be controlled by the development team based on their means, aspirations, and platform. The technical architect views feature requests through feasibility perspective to suggest changes to incorporate the new feature requests. Developers also provide technical feedback on the feasibility of features and potential avenues to improve the features. At TB, at small scale, developers are allowed to adopt an ad-hoc approach to research components which they may deem worthy. The benefit associated with such impromptu forages are identification of new and improved features, whereas tracking such endeavors is a challenge.
Our studies also highlight the dynamics of the process to identify and incorporate novel features in the application. We find that roles in the planning and visionary quadrant identified key features of the application based on their understanding of the market and technology. Effectual approach realized these ideas such that the initial ideas evolved during the effectual process.

6.4 Artifacts to Support Planning and Execution and Monitoring

Based on this research, we can identify two broad phases of the effectual software development process as seen in Figure 14—planning and execution and monitoring (Malgonde and Hevner 2016). In the planning phase, the development team adopts an effectual approach to develop an
emergent specification for the application. In the execution and monitoring phase, the development adopts an effectual approach to develop and revise the application.

**Figure 14. Two phases of Effectual Software Development Process**

The process starts with recognized opportunity. Opportunity may be recognized by a client, another department, or vendor. Identified opportunity informs aspirations of the team. These aspirations may be abstract goals for the application. Based on available means of the team and aspirations, the development team identifies possible action alternatives. Actions alternatives which have acceptable risk profile and are controllable are selected. Intermediate artifacts, effect, expands resources and attenuates aspirations. The exit condition for planning phase is a specification-aspiration match. Focusing on existing means allows the project team to control the future events for the application to be developed. Finally, aspirations shape the development of the application along with market and platform feedback. These feedback mechanisms (i.e. monitoring during execution) control the development process to maintain application-market match, application-platform match, added value proposition, and novelty of the application.
Figure 15. Flow of Planning Activities

In the Planning stage, the project manager (PM) and the development team start from an identified opportunity that provides a clear set of aspirations for adding novel customer value to the platform ecosystem. Existing means identify possible actions for the project team. Based on the risks associated with each alternative and the ability of the project team to control those risks, actions that have an acceptable risk portfolio are determined. Simultaneously, based on the current state of the platform, controls available to the project manager, and the aspirations of the project team, controllable aspects of possible actions are identified. Possible actions, that have an acceptable risk portfolio and are under the team’s control, are selected. These actions lead to effects which are the operationalization of the aspirations. When effects are realized, they provide new means in the form of knowledge, stakeholders, and possible actions – expanding the cycle of resources.
In addition, effects have a constraining effect on the project team’s aspirations. The constraining effect assists the project team to attenuate broad aspirations – leading to a converging cycle of constraints. Iteratively, when the project team achieves an effect-aspiration match, the partial specification is developed. Upon satisfactory matches between effects and aspirations, a partial specification of the desired application is produced and delivered into the *Execution* stage. A flow chart of Planning activities is shown in Figure 15.

**Figure 16. Flow of Execution and Monitoring Activities**

Upon entry into development execution, the project manager builds a portfolio of dynamic controls based on the partial application specification and existing controls in the endogenous environment (Harris et al. 2009; Kirsch 1997; Malgonde et al. 2015). The portfolio of controls enables the project manager to direct development efforts to realize a partial product. The evolving application is adjusted based on feedback from the market, platform, and team’s entrepreneurial aspirations for the product. Feedback is incorporated into the partial specification which acts as an input for
the next iteration. A key activity during execution is the monitoring of development progress and rapid feedback for adding, deleting, and/or modifying the control portfolio. For example, as questions on customer interfaces arise based on initial prototypes, the project could include additional types of user testing controls with different stakeholder groups for improved feedback. A flow chart of Execution activities is shown in Figure 16. Future research on these flows will adopt a design science approach to develop and evaluate process models. The process models will provide actionable guidance to software development teams. Such process models will consider the specific context and key aspirations of the software development team.

In this chapter, we have discussed four broader research implications of this dissertation. Future research will build on these implications by separately analyzing the data collected in this study and identify new case studies. Separate focus on each of these research implications will open interesting research areas and develop new theories for the development of software applications.
CHAPTER 7. CONTRIBUTIONS, LIMITATIONS, AND FUTURE RESEARCH

In this study, we focus on the unique challenges offered by digital platform environment for software development teams. Traditional view of technological platform differs from the digital platform considered in this research because digital platform offers generativity, market, and competition. According to the traditionally view, software application development team could select from different competing technology, market segment, and regulations. Such choice gives rise uncertainty associated with technology and market.

In a platform environment, technology and market are identified by the platform environment. While preexisting technology and market may seem to reduce uncertainty, platforms update their interfaces and functionalities. Such updates may be requested by the developer community, and/or forced on the platform owner in response to change in user preferences and competing platforms. Frequency of changes to the digital platform are fast in comparison to technological platforms.

Prior work in software development has considered the success criteria of project performance (Weiner et al. 2016) and product-market match (Harris et al. 2009a). Using portfolio of controls, software development teams balance control and flexibility required to achieve the success criteria. In this study, we identify additional success criteria for applications on digital platforms: (a) product-platform match, (b) exceed platform’s core value proposition, and (c) novelty.
Augmented success criteria for applications on digital platforms are necessary because of the uncertain, risky, and resource-constrained setting offered by digital platforms. Uncertainty in the platform environment is related to users’ preferences, platform’s architecture and governance mechanisms, and connection interfaces offered by the platform. The platform’s environment such as regulation and competition also introduce uncertainty on the digital platform. Also, risk associated with the choice of platform and technology is substantial because users move away from the platform, the application may not be relevant. Success of the application is tied to the platform. Further, the platform abstracts implementation of common functionalities and provide interfaces for applications to connect and utilize the features. However, such abstract implementations may introduce limitations on supported format, performance, or feature-set.

This study focuses on one of the key success criteria for the application—novelty. An application is novel if it provides features and extensions that the platform and other applications do not provide. Entrepreneurs “create something new, something different; they change or transmute values” (p. 22) (Drucker 2014). Thus, this study builds on entrepreneurship literature to identify approaches to develop novel applications on digital platforms.

Harris et al. (2009a) identify controlled-flexible approach to address the uncertainty of technology and market to satisfy the success criteria of product-market match. Similarly, other approaches such as plan-driven and ad-hoc approaches favor control and flexibility, respectively. In this study, we draw upon the framework of prediction and control (Wiltbank et al. 2006) to understand the underlying theoretical underpinnings of these approaches. Specifically, plan driven approaches emphasize prediction, ad-hoc approach emphasizes adaptation, and ad-hoc approaches balance prediction and adaptation. In case of applications for digital platforms, the
environment requires control-based approaches. The theory of effectuation (Sarasvathy 2001) advances an approach to control and shape the environment.

Building on the theory of effectuation, this study proposed an effectual approach to software development on digital platform. The research model is augmented with two studies—qualitative secondary data analysis of Apache Cordova and two pilot interviews at local companies. The revised research model is validated with two case studies which are developing novel software applications.

We find evidence for the effectual approach in both software development teams. We also find that the nature of application, platform’s maturity, and aspirations, play critical role in the development of novel applications. The tight effectual cycles produce intermediate artifacts which represent the current design knowledge of the team (Mullarkey and Hevner 2018). Based on contextual factors, the development team may release the application to platform. Released application provides feedback to improve the application.

7.1 Contributions

7.1.1 Theoretical Contribution

This study contributes to theory of software development. Below, we look at the criterion proposed by Whetten (1989). First, this study develops a research model of effectual software development process by extending the theory of effectuation. Extension to the theory of effectuation is in the form of new constructs and relationships. Although addition of constructs to the theoretical model helps to improve our understanding of the phenomenon, addition of constructs does not alter the core logic of the model (Whetten 1989). This research addresses this concern by discussing how the accepted relationships are altered with the introduction of new constructs. For example, introduction of platform as a construct alters the understanding of how
alternatives are identified. Similarly, effects are intermediate design artefacts which are improved to identify the final deliverable. When considered together, these constructs and relationships alter and augment our understanding of how novel software applications are developed in digital platform environments.

Second, this study contributes to the theory of software development by borrowing effectuation perspective from the entrepreneurship domain and strategic management. Specifically, we investigate the underlying theoretical underpinnings of existing software development approaches using the framework of prediction and control (Wiltbank et al. 2006). The framework allows us to reconceptualize software development approaches as strategies and assess their applicability of new domains such as digital platforms. Traditional view of software development focused on balancing control and flexibility. The effectuation perspective considered in this study allows us to consider additional perspectives of control and prediction.

Third, Whetten (1989) notes the importance of feedback loop of theoretical contribution – the study should inform us about the existing theory. This study contributes to the entrepreneurship domain empirically supporting the notion that entrepreneurs may adopt other approaches in conjunction with the effectual approach. Further, this study contributes by illustrating the approach adopted by different team members, even though the overall approach by the team is effectual.

7.1.2 Information Systems Researchers

Information systems research in software development has focused on approaches, processes, and techniques, to manage the challenges of project performance (Weiner et al. 2016) and product-market match (Harris et al. 2009a). These studies consider the uncertainties associated with the technology and environment for which the application is developed. This study
contributes to the literature in software development by (a) focusing on digital platform environment, and (b) identify new challenges faced by application development teams in platform environment—product-platform match, exceed platform’s core value proposition, and novelty.

In addition to the success criteria of product-market match, this study introduced product-platform match as another success criteria for the application development team. Prior software development studies considered technological uncertainties for the application. Once addressed, the application could standalone in the market. In a platform context, the application is required to match the platform. In other words, the application should always be compatible with the platform’s connectors via APIs and components. Thus, any changes to the application and/or platform requires attention to maintain application-platform match.

Another success criteria identified in this study is that the application’s value should exceed the core value proposition of the platform. The digital platform provides core value proposition to its users. Core value proposition is in the form of payment services, user interface and experience, and context specific features such as reporting and analytics. Often, the platform owner will introduce applications on the platform which provide basic functionality to the users. For example, Apple and Google provide applications to perform basic operations such as email, payment, word processing, notes, calling, and texting. To maintain relevance, applications need to exceed the core value proposition of the platform. For example, Microsoft offers a successful email client for Apple and Google platforms by exceeding its core value proposition via user interface and customizations.

Focusing on the novelty of the application, this research brings together ideas from the entrepreneurship and information systems domain. This research illustrates the challenges and
benefits of incorporating ideas from these domains. Entrepreneurship domain considers processes and decision-making in entrepreneurs (individuals or small group) whereas information systems consider processes and decision-making in teams with hierarchical structures and organizational constraints.

This research contributes to the information systems literature on systems development by illustrating the theoretical underpinnings of software development approaches. Specifically, building on the framework of control and prediction (Wiltbank et al. 2006), this study highlights the core strategic assumptions and logic associated with plan-driven, ad-hoc, and controlled-flexible approaches. In addition to illustrating the theoretical underpinnings of these approaches, the analysis with the framework of control and prediction also highlights the strengths, weaknesses, and assumptions of these approaches. We note that this analysis complements the differences in control portfolios of these approaches illustrated by Harris et al. (2009b).

One of the key contributions of this research is the model of effectual software development. The model augments the theory of effectuation (Sarasvathy 2001) with constructs from literature on software development and digital platforms. The model also illustrates the decision heuristics, relationships, and outcome, of the effectual software development approach. Finally, this research contributes to the software development literature in information systems by highlighting the assumptions and decision-making heuristics of different roles in a software development team. As the project progresses, we see changes in the decision-making heuristics for same role in the team.

7.1.3 Entrepreneurship Researchers

Entrepreneurship researchers have studied software development teams as entrepreneurs (Livingston 2007). Such studies focus on small teams with entrepreneurial activity in the given
application domain. This research extends the study of entrepreneurship in software development teams characterized by hierarchy, control modes, communication channels, and organizational dynamics.

Perry et al. (2012) note that one of the challenges faced by effectuation researchers is the ability to observe and measure effectuation’s constructs in action. This study illustrates that software development projects are suited to observe and measure effectuation’s constructs. Unlike individual entrepreneurial endeavours, software development projects document ideas, decision-making is visible (discussions and backlogs), outcomes are visible, and risks and control is measurable. As software development projects are managed using project management tools, archival data is available for analysis.

Sarasvathy (2001) identifies effectuation as an approach to creation of new artefacts. This research provides empirical support for effectuation as an appropriate approach to develop novel applications on digital platforms. However, this study also highlights the use of other approaches by different roles in the software development team. Further, this study contributes to entrepreneurship domain by illustrating the interplay between approaches of planning, visionary, and effectuation.

Finally, this study has identified constructs, subconstructs, and operational definitions, for effectual concepts in the context of software development. Entrepreneurship research can use these to develop measures in the context of software entrepreneurs.

7.1.4 Information Systems Practitioners

This study is one of the first to highlight the challenges of software development teams in digital platform environments. Identifying the need for novel applications on platforms, this study
proposes and evaluates an effectual approach to software development. Novel applications on
digital platforms is important for platforms as it attracts new users.

Our empirical studies highlight the assumptions and decision heuristics used by different
team members given the state of the application and their role. Understanding of these issues
enables the software development team to evaluate its current state and adapt to changing
environment. Analysis of the differing roles also enables the development team to identify team
members and alter team dynamics.

Traditionally, client/customer is identified as the entity to request an application. The
client would state the requirements and identify features which will maintain relevance of the
application after completion. This study highlights the importance of the technical team which
serve to (a) evaluate feasibility of identified features, (b) recommend alterations to identified
features based on technological capabilities, and (c) identify new features.

Finally, this research highlights the importance of platform’s maturity and complexity for
platform owners. Mature platforms allow software development teams to concentrate on the
application’s value and novelty rather than struggling with application’s match with the platform.
Also, defined schedule of changes to the platform, preceded by pre-releases and documentation
allow development teams to plan and manage changes to the application. Similarly, as digital
platforms introduce new functionalities, complexity of the platform grows and challenges the
application development team. Modular design, updated documentation, and training are some of
the channels identified in this study which can help software application development teams to
cognitively control the complexity of the platform. APPENDIX 1 discusses key differentiating
aspects of plan-driven, ad-hoc, controlled-flexible, and effectual approach.
7.2 Limitations

This study has three limitations. First, as inferences are contextualized, an inherent limitation of case study research is the generalizability of its findings beyond the context in which studied (Bhattacherjee 2012). Thus, the inferences drawn from this research may need additional analysis before applying to other context such as cloud-based software development.

Second, our case studies included teams of medium size. Applicability of the inferences drawn from this research to large software development teams may need additional analysis. Often larger teams involve fewer communication channels, bureaucratic structure, and organizational constraints. Incorporating these nuances in the effectual software development process may need additional studies. Finally, the cases considered in this research represent in-house development and client-vendor relationship. For software application projects where the development is offshored may require additional analysis.

7.3 Future Research

Entrepreneurship research has considered opportunity creation as discovery, creation, and user generated (Alvarez et al. 2013; Shah and Tripsas 2007). This research highlights the digital platform as an artefact which provides opportunity for entrepreneurial activity. Digital platforms are also characterized by resource-constraints, uncertainty, risk, and competition. Thus, the context of digital platforms may provide opportunities for future research on effectuation. Also, the software development teams marry the processes of opportunity discovery and creation, in a team setting.

This study augments the theory of effectuation with constructs and subconstructs for software development. Entrepreneurships studies may identify similar constructs and subconstructs in future research. Similarly, the model presented in this research can be further
adapted and tested in other domains of software development such as blockchain, cloud, and security.

Control theory has been employed to understand the control portfolio used by project managers in software development projects. Harris et al. (2009b) illustrate the combinations of control modes used by different approaches. Weiner et al. (2016) survey prior studies using control theory and the efficacy of portfolio of controls to meet project’s success criteria. This research can be further extended by focusing on the portfolio of controls used by project managers in the development of novel applications on digital platforms. Future studies may consider case studies of projects to illustrate the configuration, dynamics, and enactment, of control modes in such projects.

In this research, we find that culture of the software development team plays a key role. Specific characteristics such as open communication channels, lack of hierarchy, and open evaluation of emerging ideas, allow the team to enumerate effectual cycles. Culture facilitates evaluation of ideas which may be identified by springboard customers. Also, culture facilitates nurturing nascent ideas from team members. For example, emergent ideas are identified in the project backlog and revisited during grooming sessions. Future research can consider the ideas which influence the team’s culture.

This research focused on the challenges faced by applications producers in the platform ecosystem. Future research can extend this research to consider the challenges faced by platform owners and incorporate effectual thinking to maintain platform’s relevance. As competing platforms and environment evolve, platform owner(s) need to change architecture and governance mechanisms of the platform. These changes, in turn, affect the connection interfaced offered by the platform. Further, complexity and maturity of the platform influence producers
action possibilities. Future research can design processes to augment such decisions such that its
effect on producers is streamlined.

Finally, the framework on prediction and control identified four quadrants which identify
underlying theoretical underpinnings of software development approaches. The effectual
approach is appropriate in environments which represent the transformative quadrant. Other
challenging environments such as Platform which represent the visionary quadrant require future
research to identify software development approaches suitable for the challenges in visionary
quadrant.

7.4 Conclusion

This study identified key challenges for software development team in the context of digital
platforms and extend the success criteria for applications on digital platforms: product-market
match, product-platform match, exceeding platform’s core value proposition, and novelty.
Building on the framework of control and prediction, we find limited applicability of existing
approaches to address the challenges and meet the success criteria.

With the theoretical basis of the theory of effectuation, this study proposed an effectual
approach to software development. Preliminary support for effectual thinking is provided by
secondary data analysis from Apache Cordova’s stories. The preliminary support is further
augmented by two pilot interviews. Results from data analysis of the two pilot interviews
identify new constructs and subconstructs to augment the model of effectual software
development.

To validate the model, we conduct two case studies. We find support for the efficacy of
effectual approach to develop novel applications on digital platforms. We also find that different
roles assume different strategies during the development process. Finally, although novel ideas are identified with planning and visionary approaches, effectual approach enables the team to evaluate alternatives and realize the ideas.
REFERENCES


This appendix discusses key differentiating aspects of effectual approach from controlled-flexible approaches. Further, we illustrate and discuss key aspects of the proposed effectual approach to existing software development approaches. The comparison allows us to understand the key differentiators, strengths, and limitations of these approaches.

The effectual approach shares some of the practices of the controlled-flexible approach. However, the effectual approach extends our thinking along the lines of success criteria, decision heuristics, and project execution. First, controlled-flexible approaches focus on product-market match. Effectual approach extends this success criteria by including novelty of the application which is important in an environment characterized by uncertainty, risk, and resource constraints. Second, controlled-flexible approaches emphasize iterative development to allow flexibility. Each cycle/iteration allows the team to evaluate the product-market match and realign, if needed. An effectual approach also emphasizes iterative development. Similar to controlled-flexible approach, each iteration/cycle allows the team to evaluate the product with the success criteria. However, the effectual approach focuses on building and assimilating the new knowledge to expand its resources and attenuate its aspirations. For controlled-flexible approach, the key evaluation criteria is the product-market match. In effectual approach, the team’s aspirations (generalized end goals) form the key evaluation criteria before the application is deployed. Third, controlled-flexible approach evaluates action alternatives based on the logic of expected returns. In other words, action alternatives are favored if they provide the greatest return to achieve the success criteria. The effectual approach evaluates alternatives based on their
risk and controllable aspects. In other words, action alternatives are chosen if the risk associated with those actions are acceptable to the team and if all the aspects of the action alternative are controllable to the team. Finally, controlled-flexible approach identify intermediate working product as an iteration’s product which provides feedback to subsequent iterations. In effectual approach, fast effectual cycles identify design artifacts (documentation, code, prototypes, proof of concepts, among others) as intermediate effects to provide feedback to subsequent iterations.

Table 13. Software Development Approaches

<table>
<thead>
<tr>
<th>Assumption (Environment)</th>
<th>Plan-driven</th>
<th>Controlled-Flexible</th>
<th>Ad-hoc</th>
<th>Effectual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption (Market)</td>
<td>Mature market</td>
<td>Mature but fast evolving market</td>
<td>Mature but fast evolving market</td>
<td>Nascent market, mature but fast evolving</td>
</tr>
<tr>
<td>Project Execution</td>
<td>Linear</td>
<td>Iterative</td>
<td>Iterative</td>
<td>Iterative (effectual cycles)</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Prediction, resource gathering, milestones</td>
<td>Partial prediction, resource gathering, scope boundaries, ongoing feedback</td>
<td>constant calibration of its course by reacting to changes in the environment</td>
<td>Existing means, aspirations, acceptable risk, control, effects</td>
</tr>
<tr>
<td>Underlying Logic</td>
<td>Causation</td>
<td>Causation and adaptability</td>
<td>Rapid Adaptation</td>
<td>Effectuation</td>
</tr>
<tr>
<td>Process</td>
<td>Define outcome (specification) based on consumer needs, gather required resources, plan milestones,</td>
<td>Define partial outcome based on consumer needs, gather required resources, set scope boundaries,</td>
<td>Development follows changes in consumer needs</td>
<td>Define partial outcome using aspirations, existing means, and consumer needs, identify multiple effects from existing</td>
</tr>
<tr>
<td>Decision making</td>
<td>Systematic information gathering and analysis, expected return of the alternative</td>
<td>Systematic information gathering and analysis, expected return of the alternative, feedback</td>
<td>Expected return</td>
<td>Acceptable risk and logic of control</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Outcome</td>
<td>Product-specification match</td>
<td>Product-market match</td>
<td>Working product</td>
<td>Product-market match, product-platform match, value exceeding platform’s core proposition, novelty</td>
</tr>
<tr>
<td>Use of Knowledge</td>
<td>Rely on existing knowledge base</td>
<td>Synthesize existing and new knowledge (created with every iteration)</td>
<td>Act on new situational knowledge</td>
<td>Synthesize existing and new knowledge (created with every effect)</td>
</tr>
</tbody>
</table>

The effectual approach to software development is suitable to uncertain, risky, and resource constrained environments. Suitability of effectual approach to such an environment is due to the control-based approach of the effectuation process. Other approaches address uncertainty with rapid adaptation. Effectual approach advocates emphasis on existing means and aspirations to identify action alternatives with acceptable risk and controllable aspects.
Another key differentiator in the effectual approach is the intermediate effects which embody the design knowledge of the team and facilitate evaluation. Intermediate effects are artifacts which may be designs, backlog items, documentation, builds, proof of concepts, among others. These tangible artifacts allow the development team to evaluate ideas, identify new resources, and attenuate the team’s aspirations.

Effectuation focuses on controlling rather than predicting. In the software development space, other approaches focus on expected return of alternatives whereas effectual approach focuses on controllable alternatives. In this research, we also find that effectual approach balances prediction and control across different roles within the team.
### APPENDIX 2  CODING EXAMPLES FOR APACHE CORDOVA

The stories in the table below provide examples of the coding performed in the qualitative data analyses.

<table>
<thead>
<tr>
<th>Story Description</th>
<th>Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>with cordova-plugin-contacts 1.1.0 &quot;Contacts&quot;</td>
<td>Acceptable Risk - Commit limited resources</td>
</tr>
<tr>
<td>When I get the whole list of contacts It's working but when I choose:</td>
<td>Action – Fixed bugs</td>
</tr>
<tr>
<td>navigator.contacts.fieldType.phoneNumbers in the options.desiredFields</td>
<td>Expanding cycle of resources – new technical knowledge</td>
</tr>
<tr>
<td>I get properly some first contacts, then (maybe because one of my contact</td>
<td>Means – Technology</td>
</tr>
<tr>
<td>phoneNumber value) , I get this error :</td>
<td></td>
</tr>
<tr>
<td>Error in Success callbackId: Contacts598408154 :</td>
<td></td>
</tr>
<tr>
<td>SyntaxError: Unexpected token u</td>
<td></td>
</tr>
<tr>
<td>cordova.js:312</td>
<td></td>
</tr>
<tr>
<td>Cordova plugin contacts - PhoneNumbers error</td>
<td></td>
</tr>
<tr>
<td>Add the file plugin and browser platform (edge, from github or local repo)</td>
<td>Action – Fixed bugs</td>
</tr>
<tr>
<td>then cordova run browser gives the following in the console:</td>
<td>Expanding cycle of resources – new technical knowledge</td>
</tr>
<tr>
<td>Error: exec proxy not found for :: File :: requestFileSystem</td>
<td>Platform - Technology</td>
</tr>
<tr>
<td>File plugin on browser platform causes &quot;proxy not found&quot; error on Chrome</td>
<td></td>
</tr>
<tr>
<td>Most of automatic geolocation tests were pended on Android because we didn't</td>
<td>Means, Action, Aspiration – Novelty, application-platform match, Logic</td>
</tr>
<tr>
<td>have the tool to detect if the tests are running on a simulator or on a real</td>
<td>of Control</td>
</tr>
<tr>
<td>device. Now we have device.isVirtual and can use it to pend the tests only on</td>
<td></td>
</tr>
<tr>
<td>an emulator. Make geolocation tests use device plugin to properly detect Android</td>
<td></td>
</tr>
<tr>
<td>simulator</td>
<td></td>
</tr>
<tr>
<td>I came back from the Android Dev Summit, and sure enough, I forgot about the &quot;Do</td>
<td>Acceptable Risk, Aspirations, Expanding cycle of resources,</td>
</tr>
<tr>
<td>not show me again&quot; box on permissions. We need to handle this somehow and send a</td>
<td></td>
</tr>
<tr>
<td>a different exception. We should</td>
<td></td>
</tr>
</tbody>
</table>
allow developers to tell users why they need the permission, otherwise their application experience will suffer. This will be an API addition, not a change. I don't believe we need to go up a major version for this. Cordova does not handle use case where we need to show rationale about permissions

converging cycle of constraints, means, platform
APPENDIX 3  ILLUSTRATIVE EXEMPLARS OF APACHE CORDOVA

In this appendix, we present and discuss several illustrative exemplars of qualitative secondary data from the Apache Cordova platform repository and the constructs identified in the effectual software development model.

Means

Means represent the overall existing knowledge about the current state of the application, platform, market, and social capital that the project team draws upon. Means for the development team consists of software development kits, documentations, discussion boards, mailing lists, and so on. These resources provide a set of means that are collectively identified and referred to by the team to generate alternatives for actions that develop intermediate effects. The qualitative data available in this analysis consists of story descriptions of issues and features identified by the contributors. This leads us to means that are not explicitly stated in the descriptions but implied in the discussions. For example, consider the following description of a story:

Under Adobe AIR, you can open a connection to a SQLite db and point to an existing file. The benefit of this is that your application can ship a database seeded with data. Without this support, your application has to initialize the db via scripting. While not difficult, it does increase the application's first run time and also complicates the code unnecessarily. I understand that this isn't per the Web SQL spec, http://dev.w3.org/html5/webdatabase, but it could certainly be useful.
The contributor is discussing a feature that is introduced in the application. The technological and platform-specific means posed by the contributor identifies this enhancement and the team relies on its means (tools, programming language, design, architecture, platform interface, and market) to evaluate possible alternatives and introduce it in the application. Specifically, the contributor identifies a specific plugin that enhances value to the existing framework. The risks associated with standards (W3C) are also discussed. Consider another example of means-driven approach as illustrated in the following story description, where tools are identified to develop test cases.

Most of automatic geolocation tests were pended on Android because we didn't have the tool to detect if the tests are running on a simulator or on a real device.

Now we have device.isVirtual and can use it to pend the tests only on an emulator.

Platform

The Platform is the centerpiece around which decisions and choices are made for the application. Platforms provide and constrain the application development context. In the following story, the contributor identifies a specific framework in the iOS platform. Demonstrating reusability and modularity of the platform design, the framework is used across multiple plugins within the platform (as listed by the contributor). However, it constraints application developers because updates to platform components may often require significant change to the application.

The ALAssetsLibrary framework has been deprecated in iOS 9, replaced by the Photos.framework. Once our minimum dependency is iOS 9, move to it. Usage: 1. iOS (CDVURLProtocol); 2. Camera plugin; 3. File plugin; 4. File Transfer plugin; 5. Local-Webserver plugin (cordova-plugins)
Aspirations

The aspirations of ‘product-market match’ and ‘product-platform match’ are implicit in team’s actions. It follows that the application design and development should ultimately ensure that the application works with the platform. Also, feature requests are accompanied with the limitations of the platform’s value proposition and the added value proposed by the contributor. As an illustration, consider the following story where the contributor identifies (a) the value provided by the platform (Android and iOS), (b) a platform-market need that is not satisfied and subsequently the value that is added through this feature, and (c) using technological and platform means, possible actions are suggested for both of these platforms.

The use case is when an app/user needs to access geoposition while device’s location services are disabled. Let's say for the first time / attempt. While I've been able to find a way to send the user directly to the system setting on Android (via cordova-diagnostic-plugin's switchToLocationSettings), it seems to be no obvious way to achieve the same on iOS with the plugin(s) at present. ... I thus suggest extending getCurrentPosition with an option for a better UX in case the device’s location services are disabled. ... I would suggest covering the same for Android, even though this issue is concerned mainly with the UX on iOS.

Acceptable Risk

Feature requests and issues are accompanied with risk analysis. Typical areas of risk analysis include identification of alternatives—technological, platform, and/or market, risk associated with the alternatives, and the resources required to realize the alternatives that have been identified. Consider the following story description (listing added) where the contributor identifies an issue in dynamic programmatic calls for specific platform. The issue is identified,
elaborated, and alternatives are discussed. Finally, the committer narrows to a specific plan of action.

We have a logic in Windows/wp8 parsers that fires a hooks, specific for these particular platforms. There is some problems with this: (a) This doesn't fits well into the concept of PlatformApi (b) The original purpose of the hook is now lost. ... So the proposed plan is: (a) Do not touch 'pre_package' if 'old' platform is used (via PlatformApi polyfill); (b) If the 'new' platform is used, 'pre_package' doesn't emitted by platform, so we need to emit it manually (right before 'after_prepare' - to keep the order of hooks unchanged); (c) Move bomify from prepare to build in Windows PlatformApi, so www sources will be not-yet-bomified in 'pre_package'; (d) Add a notice about 'pre_package' deprecation and removal to HookRunner

Logic of Control

With logic of control, the project team is selecting actions that they have control over rather than predicting if and when the features and/or issues will be identified and resolved on the platform. Story descriptions do not speculate on the possible directions in which platforms will change. Rather, alternatives are identified based on the means and aspirations of the team. Consider, for example, the following story which discusses an issue with two platforms. Relying on the means (technological and platform) and the knowledge about platform leads to the identification of this issue. Instead of reporting the issue to platform and waiting for a fix in its next version, the contributor has provided a fix and tested it across multiple devices.

MediaFile.getFormatData result data was empty (filled with default "0" values) for all types of capture: image, video audio. Problem encountered on Android iOS. I solved this by changing the url passed to native code from localURL to
Actions and Effects

All stories within our dataset are marked as complete because the issue/request has been resolved. These completed stories represent the actions of the project team to generate intermediate effects in the project. In the Cordova project, effect, which is the operationalization of team’s aspirations, is the collective documentation of which features and issues are to be addressed in the Cordova project. Each intermediate release of the project represents an effect for the overall effectuation process in the Cordova project which converges constraints and expands means. Consider another story’s description from our dataset. The story is discussing a flexible cropping feature unavailable on iOS platform (an effect that provides novelty, platform-market match, and value to that provided by the platform) for pictures.

On iOS there's only that very insufficient inflexible cropping square compared to Android or WinMobile which moreover obviously doesn't work properly (see CB-9930, CB-2648). As we need a flexible, sizable rectangle, we implemented that in our fork of the camera plugin. ... To be downward compatible and to not urge others, for whom that square may be sufficient, it is made parametrizable via a new preference (as this is iOS specific and nothing that has to be changeable at runtime), defaulting to false. If the plugin is called with option allowEdit == true, then setting this new preference to true suppresses that standard (fairly useless) square for cropping the photo, even suppresses the (then also useless) view of the photo with the "Retake"- and "Use Photo"- buttons, but instead offers a resizable cropping rectangle (with "Redo" and "Save". ...
As the effect is identified, project teams converge constraints on application design, technology, and platform match. Similarly, these intermediate effects lead to identification of additional effects, and technological, platform-specific, and application-specific means to the team.
## APPENDIX 4  INTERVIEW QUESTIONNAIRE

Interview Questionnaire – Project Manager

<table>
<thead>
<tr>
<th>Question Number</th>
<th>Question</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Describe your job role in the project.</td>
<td>Background information on the platform ecosystem and application to describe, understand, and triangulate the case.</td>
</tr>
<tr>
<td>2</td>
<td>Describe the platform ecosystem for the application.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Describe the application domain.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Describe the development process followed in general.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Who are the applications users and how does this application provide value to them?</td>
<td>Novel and value adding components/features of the application</td>
</tr>
<tr>
<td>6</td>
<td>What are some of the components of the application that sets it apart from existing applications on the platform? What is different about it?</td>
<td>Were the novel components identified a priori? – prediction based logic</td>
</tr>
<tr>
<td>7</td>
<td>Is this application novel? How?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Who identified some of these novel components of the application?</td>
<td>Important and key aspiration(s) for the application</td>
</tr>
<tr>
<td>9</td>
<td>Were they identified by the user?</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Were they known a priori to the project? Why?</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>What was the most critical criteria for the success of this application?</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>These are some of the broader criteria (aspirations) for application development on platforms (application-platform match, application-market match, value exceeding platform, novelty). Do you have any other category of success criteria (aspiration) for your application? Why?</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>In an iteration, do you focus on a specific broader success criteria (aspiration)? Or is it a mix of different success criteria (aspirations)? Why?</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Who and how did they identify these novel components in the application?</td>
<td>Means to identify novel components</td>
</tr>
<tr>
<td>15</td>
<td>What was your role in identifying these novel components?</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>What was the key source of information and knowledge that they relied on to identify these novel components?</td>
<td></td>
</tr>
<tr>
<td>Question Number</td>
<td>Question</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Describe your job role in the project.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Describe the platform ecosystem for the application.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Describe the application domain.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Describe the development process followed in general.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Who are the applications users and how does this application provide value to them?</td>
<td>Novel and value adding components/features of the application</td>
</tr>
<tr>
<td>6</td>
<td>What are some of the components of the application that sets it apart from existing applications on the platform? What is different about it?</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Is this application novel? How?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Who identified some of these novel components of the application?</td>
<td>Were the novel components identified a priori? – prediction based logic</td>
</tr>
<tr>
<td>9</td>
<td>Were they identified by the user?</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Were they known a priori to the project? Why?</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>What was the most critical criteria for the success of this application?</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>These are some of the broader aspirations for application development on platforms (application-platform match, application-market match, value exceeding platform, novelty). Do you have any other category of aspiration for your application? Why?</td>
<td>Important and key aspiration(s) for the application</td>
</tr>
<tr>
<td>13</td>
<td>In an iteration, do you focus on a specific broader aspiration? Or is it a mix of different aspirations? Why?</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Who and how did they identify these novel components in the application?</td>
<td>Means to identify novel components</td>
</tr>
<tr>
<td>15</td>
<td>What was your role in identifying these novel components?</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>What was the key source of information and knowledge that they relied on to identify these novel components? Was it specific—technology, platform, market, users; or draws on multiple factors?</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>How were the novel components incorporated into the application over time? Was the idea of novel components honed to incorporate in the application or implemented as soon as identified?</td>
<td>Feedback</td>
</tr>
<tr>
<td>18</td>
<td>How do intermediate iterations help to identify novel components in the application?</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>How do you know a component/feature is ready for development?</td>
<td>Artifact</td>
</tr>
<tr>
<td>20</td>
<td>How do you decide if a component or feature should be incorporated in the application?</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Did intermediate iterations increase your knowledge about the application? How?</td>
<td>Expanding cycle of resources</td>
</tr>
<tr>
<td>22</td>
<td>Did intermediate iterations help you identify specific goals in subsequent iterations? How?</td>
<td>Converging cycle of constraints</td>
</tr>
<tr>
<td>23</td>
<td>Discuss the process outlined in the model to identify novel and value adding software component that can be developed.</td>
<td>Research Model</td>
</tr>
</tbody>
</table>
## APPENDIX 5  REVISED INTERVIEW QUESTIONNAIRE

### Interview Questionnaire – Project Manager

<table>
<thead>
<tr>
<th>Question Number</th>
<th>Question</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Describe your job role in the project.</td>
<td>Background information on the platform ecosystem and application to describe, understand, and triangulate the case.</td>
</tr>
<tr>
<td>2</td>
<td>Describe the platform ecosystem for the application.</td>
<td>Novel and value adding components/features of the application</td>
</tr>
<tr>
<td>3</td>
<td>Describe the application domain.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Describe the development process followed in general.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Who are the applications users and how does this application provide value to them?</td>
<td>Novel and value adding components/features of the application</td>
</tr>
<tr>
<td>6</td>
<td>What are some of the components of the application that sets it apart from existing applications on the platform? What is different about it?</td>
<td>Were the novel components identified a priori? – prediction based logic</td>
</tr>
<tr>
<td>7</td>
<td>Who identified some of these novel components of the application?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Were they identified by the user?</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Were they known a priori to the project? Why?</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>What was the most critical criteria for the success of this application?</td>
<td>Important and key aspiration(s) for the application</td>
</tr>
<tr>
<td>11</td>
<td>These are some of the broader criteria (aspirations) for application development on platforms (application-platform match, application-market match, value exceeding platform, novelty). Do you have any other category of success criteria (aspiration) for your application? Why?</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>In an iteration, do you focus on a specific broader success criteria (aspiration)? Or is it a mix of different success criteria (aspirations)? Why?</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>How did they identify these novel components in the application?</td>
<td>Means to identify novel components Platform’s role</td>
</tr>
<tr>
<td>14</td>
<td>What was your role in identifying these novel components?</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>What was the key source of information and knowledge that they relied on to identify these novel components? Was it specific—technology, platform, market, users; or draws on multiple factors?</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>How does the platform help to identify these novel components? How does it constraint and provide resources?</td>
<td></td>
</tr>
<tr>
<td>Question Number</td>
<td>Question</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>17</td>
<td>How were the novel components incorporated in the application over time? Was the idea of novel components honed to incorporate in the application or implemented as soon as identified?</td>
<td>Feedback</td>
</tr>
<tr>
<td>18</td>
<td>How do intermediate iterations help to identify novel components in the application?</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Do you consider the risk of incorporating these novel features? How? Why?</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>How do you know a component/feature is ready for development?</td>
<td>Artifact (effectuation cycles are complete)</td>
</tr>
<tr>
<td>21</td>
<td>How do you decide if a component or feature should be incorporated in the application?</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Did intermediate iterations increase your knowledge about the application? How?</td>
<td>Expanding cycle of resources</td>
</tr>
<tr>
<td>23</td>
<td>Did intermediate iterations help you identify specific goals in subsequent iterations? How?</td>
<td>Converging cycle of constraints</td>
</tr>
<tr>
<td>24</td>
<td>Discuss the process outlined in the model to identify novel and value adding software component that can be developed.</td>
<td>Research Model</td>
</tr>
</tbody>
</table>

**Interview Questionnaire – Team Member**

<table>
<thead>
<tr>
<th>Question Number</th>
<th>Question</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Describe your job role in the project.</td>
<td>Background information on the platform ecosystem and application to describe, understand, and triangulate the case.</td>
</tr>
<tr>
<td>2</td>
<td>Describe the platform ecosystem for the application.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Describe the application domain.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Describe the development process followed in general.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Who are the applications users and how does this application provide value to them?</td>
<td>Novel and value adding components/features of the application</td>
</tr>
<tr>
<td>6</td>
<td>What are some of the components of the application that sets it apart from existing applications on the platform? What is different about it?</td>
<td>Were the novel components identified a priori? – prediction based logic</td>
</tr>
<tr>
<td>7</td>
<td>Who identified some of these novel components of the application?</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Were they identified by the user?</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Were they known a priori to the project? Why?</td>
<td>Important and key aspiration(s) for the application</td>
</tr>
<tr>
<td>10</td>
<td>What was the most critical criteria for the success of this application?</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>These are some of the broader aspirations for application development on platforms (application-platform match,</td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>Answer</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>application-market match, value exceeding platform, novelty). Do you</td>
<td>In an iteration, do you focus on a specific broader success criteria</td>
<td></td>
</tr>
<tr>
<td>have any other category of aspiration for your application? Why?</td>
<td>(aspiration)? Or is it a mix of different success criteria (aspirations)?</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Why?</td>
<td></td>
</tr>
<tr>
<td>Who and how did they identify these novel components in the application?</td>
<td>Means to identify novel components</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Platform’s role</td>
<td></td>
</tr>
<tr>
<td>What was your role in identifying these novel components?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What was the key source of information and knowledge that they relied</td>
<td>How does the platform help to identify these novel components? How</td>
<td></td>
</tr>
<tr>
<td>on to identify these novel components? Was it specific—technology,</td>
<td>does it constrain and provide resources?</td>
<td></td>
</tr>
<tr>
<td>platform, market, users; or draws on multiple factors?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How does the platform help to identify these novel components? How</td>
<td></td>
<td></td>
</tr>
<tr>
<td>does it constrain and provide resources?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How were the novel components incorporated in to the application over</td>
<td>How do intermediate iterations help to identify novel components in</td>
<td></td>
</tr>
<tr>
<td>time? Was the idea of novel components honed to incorporate in the</td>
<td>the application?</td>
<td></td>
</tr>
<tr>
<td>application or implemented as soon as identified?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do you consider the risk of incorporating these novel features? How?</td>
<td>How do intermediate iterations help to identify novel components in</td>
<td></td>
</tr>
<tr>
<td>Why?</td>
<td>the application?</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How do you know a component/feature is ready for development?</td>
<td>How do intermediate iterations increase your knowledge about the</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>application?</td>
<td></td>
</tr>
<tr>
<td>How do you decide if a component or feature should be incorporated in</td>
<td>Did intermediate iterations increase your knowledge about the</td>
<td></td>
</tr>
<tr>
<td>the application?</td>
<td>application?</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Expanding cycle of resources</td>
<td></td>
</tr>
<tr>
<td>Did intermediate iterations help you identify specific goals in</td>
<td>Did intermediate iterations help you identify specific goals in</td>
<td></td>
</tr>
<tr>
<td>subsequent iterations? How?</td>
<td>subsequent iterations?</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Converging cycle of constraints</td>
<td></td>
</tr>
<tr>
<td>Discuss the process outlined in the model to identify novel and value</td>
<td>Research Model</td>
<td></td>
</tr>
<tr>
<td>adding software component that can be developed.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 14. AT Case Findings

<table>
<thead>
<tr>
<th>Construct</th>
<th>First Cycle Code</th>
<th>Case Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Means (existing resources at hand)</td>
<td></td>
<td>• Microsoft windows-based architecture using csharp.net, Database: sequel server using Entity Framework, Azure computing;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Xamarin is a wrapper around the native iOS components so there are two layers of abstraction. Recently bought out by Microsoft – constantly evolving – best tool for developing iPhone apps;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Agile methodology using sprints, brainstorming and backlog/backlog grooming;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• chaining different experiences; extensive with Microsoft but have to use non-Microsoft tools too;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Use of prototyping to make sure the complex set of tools work together;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Challenge from need to use the native IOS for video and time requirements for delivery;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Team’s knowledge about Xamarian and iOS capabilities;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• rapid prototyping tools with Sketch;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• sandbox environment setups to test ideas;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• develop calls to sync with existing content infrastructure;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• advanced video player capabilities</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>• need to match to needed understanding of business context;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• research up-front on the business, users, and any competing products;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• View of the business users and clients: “it’s a collaborative process;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the client is a partner;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Feedback from product owners;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• performance expectations were high;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Time pressure to understand user requirements: From developer’s standpoint – not enough time (3 months) to really develop the business knowledge;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• few competing products for physician education so couldn’t compare to be sure to exceed those;</td>
</tr>
</tbody>
</table>
| Platform knowledge | • thought leadership on how to improve their product focusing it from the business perspective, that is more important than the technology itself;  
• Team’s ability to recommend continuous scrolling:  
| | • high frequency changes in the mobile environment and see if (a) this changes what has already been done and (b) there are now new functions that could be of use to the business;  
• syncing multiple platforms for users;  
• Importance of conforming to platform standards, and to use user controls that are consistent with what users are used to with the platform in general;  
• Culture is to constantly do personal research on what the platforms can do – and ease of sharing;  
• lot of research to see what’s coming up next;  
• Need to plan to iterate as the platform(s) change over time, and they change fast “it’s always a moving target”; abide on release;  
• scalable for different platforms (e.g. for size in mobile platforms – iPad vs. iPhone);  
• the actual platform allows us to do more, create more in a way;  
• Difficulty from building on platform without losing performance or maintainability; take these new tools and apply them to a business.”;  
• Video players; reasons for rejection of application from platform; available API’s and functionalities  
| Social Capital | • Important to the success of the product than technology issues;  
• Team needs to include technical architect, Business Analyst, product owners, developers, user interface designer; Team experience; multiple sources;  
• combination of innovative ideas from the business as well as the feedback that they receive from the users;  
• trusting us to be able to create this application;  
• Strategic thinking; Lookup information on forums such as Stackoverflow  
| Culture | • Frequent discussions  
• Open communication channels  
| Platform (resources and constraints provided) Technology (API) | • Platform is iOS + Xamarin + the cloud;  
• no limitations so far;  
• teach the business that in order to make some decisions;  
• Potential conflict between platforms;  
• Xamarin provides wrappers around iOS components;  
• easy development; limited video features;  

<table>
<thead>
<tr>
<th>Aspirations (all these aspirations are “intrinsic” to the company; all are important)</th>
<th>Market</th>
<th>Value</th>
<th>Maturity</th>
<th>Complexity</th>
<th>Product-market match</th>
<th>Product-platform match</th>
<th>Exceed Platform Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changing functionality of platform causes problems; iOS provides access to 80% of the physicians at client</td>
<td>Platform success = application match with market + app match with platform + exceeding platform value + novelty (exceeding value of competing apps); Conformity to platform standards and user interface commands makes the app user friendly; Synching across platforms; Value is always the key assessment; Choice of platform; Value assessment after product delivery; Look and feel of iOS application is better than web-based application</td>
<td>Platform is mature</td>
<td>Frequent changes to incorporate changes in consumer preferences</td>
<td>Platform is complex due to multiple components and related changes</td>
<td>the business, the client, they have the final call; lot of research up front on my client's business; knowledge base … based in a mobile medium; What the developing company can add is to do more than meet market needs, but to also think strategically what will be needed and supportable in the future; It's a project mindset, it's a myopic view and it is more operational in nature than it is strategic in nature; Challenge from designing for small screen; Mobile experience for web-based content</td>
<td>Had to use the M video player despite limitations (lag time); mobile application development is still very new; Project leaders manage expectations; Hard to explain to the clients about how the platform drives things; following standards and protocols in terms of iPhone applications; Dealing with constant platform change; more latitude to develop; scalable for different platforms; iOS should not reject the application</td>
<td>video player perspective, we did beat that thing into submission to get it to do things that it's not normally able to do; tweak it to make it get exactly what you want;</td>
</tr>
</tbody>
</table>
| Novelty | • go above and beyond to create something that made it even more convenient;  
| | • so many new things that we can do based on how iOS 11 has been updated;  
| | • Need to bring clients along as you create additional functionality;  
| | • Any tweaks made to platform’s offering (e.g. change button appearance, scrolling)  
| Novelty | • Novelty from the content and not the app; Unsure if there is novelty;  
| | • ability to copy from transcription of the video into notes;  
| | • user experience is what really takes it apart from others;  
| | • rapid prototyping was one of the key success factors;  
| | • architecture; interface using “newer styles and techniques”;  
| | • this is a great idea but do I really need it?;  
| | • ensure that there are some extra tools or tricks;  
| | • Novelty is the last consideration;  
| | • some nice-to-haves and maybe novelty items.  
| | • This would be super cool if we could do that;  
| | • to make sure it makes sense if we going to put this extra effort into creating custom elements;  
| | • Application’s ability to show content on iPhone;  
| | • track users’ progress in a video;  
| | • make notes for videos;  
| Acceptable Risk | • Primary resource to consider is the budget;  
| | • Another view on priority: “My concerns are timeline, budget, I mean we've got a scope”;  
| | • Need to simultaneously look at individual features and the whole application picture as the product is developed and decisions are made;  
| | • Risk from making a decision to add a feature;  
| | • Risk during development; security; complexity and dependencies;  
| | • When there are multiple versions of a platform in use, they consider how many users are in which version to decide which platform to design for;  
| | • During development refactoring of the code; initial planning;  
| | • Team members work on stories; different technology layer has different team member;  
| | • time box exploratory events;  
<p>| Application recoverable after failure | • it has to be long-term maintainable; |</p>
<table>
<thead>
<tr>
<th>Action</th>
<th>Fixed bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulators;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Risk Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Risk mitigation by doing the prototypes in a sandbox where they try to break it;</td>
</tr>
<tr>
<td>• At the development of each increment they consider whether the approach (like saving every x seconds) will break the application; consider security; Trial and error;</td>
</tr>
<tr>
<td>• Risk of complexity from need to use multiple tools;</td>
</tr>
<tr>
<td>• Testing not thorough enough is the biggest risk;</td>
</tr>
<tr>
<td>• Backlog refining sessions determine if a component has acceptable risk;</td>
</tr>
<tr>
<td>• Risk mitigation via the use of rapid prototyping of the core elements to see if they work; if there's a dependency on someone else to get it done;</td>
</tr>
<tr>
<td>• Risk from the platform standards or rules;</td>
</tr>
<tr>
<td>• When considering new features to add have to assess whether or not they will conflict with something already built; M-Player versus HTML player</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logic of Control</th>
<th>Logic of Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>• the logic of control in any software development really is time and a budget.</td>
<td></td>
</tr>
<tr>
<td>• It really goes into how to prioritize the action;</td>
<td></td>
</tr>
<tr>
<td>• Sprint reviews are control mechanisms;</td>
<td></td>
</tr>
<tr>
<td>• Need visual artifacts for review by product owners to improve understanding;</td>
<td></td>
</tr>
<tr>
<td>• Some things are beyond our control; never wait to build something;</td>
<td></td>
</tr>
<tr>
<td>• Advantage of a small team is that idea generation is easier to control;</td>
<td></td>
</tr>
<tr>
<td>• Trust between the development team and product owners makes it easier;</td>
<td></td>
</tr>
<tr>
<td>• Smaller projects are easier to control than larger projects because there are fewer people involved that have to agree to decisions;</td>
<td></td>
</tr>
<tr>
<td>• During design the platform is always a controlling factor; backlog of refining session; constant prioritization based on business value, based on development time and based on the availability of the technology; considering a design element have to consider if it is supportable;</td>
<td></td>
</tr>
<tr>
<td>• Need to design to be able to scale to other platforms;</td>
<td></td>
</tr>
<tr>
<td>• Need to understand future platform developments;</td>
<td></td>
</tr>
<tr>
<td>• Who controls; Backlog refining sessions focus on a story’s development, integration, testing, and deployment;</td>
<td></td>
</tr>
<tr>
<td>• build on M player because base architecture of the application did not support HTML5</td>
<td></td>
</tr>
<tr>
<td>Completed Tasks</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>---</td>
</tr>
</tbody>
</table>
| • Testing not thorough enough is the biggest risk;  
• Testing process leads to new ideas for improvement; any bugs that are found are moved into the next sprint |   |
|   |   |
| Effects | NA |
| • Means of communication;  
• Development team meetings;  
• Continuous cycle of development and delivery; strategic understanding;  
• cultural transition needed;  
• the business, the client, they have the final call; “Once we feel we’ve answered all the questions, and the stories written then we think it's ready for development.”;  
• build a relationship with the client;  
• rapid prototyping;  
• Intermediate releases after sprints |   |
|   |   |
| Expanding Cycle of Resources |   |
| • learned a lot about how the iOS delegates work and how to capture all this kind of stuff;  
• “understanding blossoms”;  
• feedback; “It's removing barrier after barrier[until] the apps perfected.”;  
• from what they offer and extend it to make it match what we need.”  
• View that no new knowledge was developed because used standard components;  
• need to do some discovery; change my approach; pick technology up on the go;  
• learning experience; “The more we had to experience, the more you'd iterate and the more you'd work on it, the more knowledge I would gain;  
• When we get new to get new hardware and new software that triggers some new ideas because now you can do certain things with this new set of tools.”  
• “Every single day, I download a new app, and learn from it. That allows me to, one, learn what exist, what new patterns are coming in, see how something works and doesn't work. “Appropriate approach to develop component; reinforces existing knowledge |   |
| New market knowledge |   |
| • Feedback on product release; new updates are coming;  
• knowledge starts growing that way if you're working on it for a longer period of time; they had a prototype coming into this of what they thought their vision would |   |
be. Of course, that evolved; the gap between development and the business is there and there’s a bridge that's been formed; constant change;

- The goal is to think for the user; 80-85% of ideas set from the beginning, and rest come during development;
- Anyone can learn the technology but if you can't figure out how to apply it to client business then it's useless to them;
- You're never going to get it right the first time; business road map look like six months out, nine months out;
- in the mobile space as frequent as operating systems get updated with new features and functionality;
- The majority of it is organic; team needs to deliver on what we've committed;
- Usage data identifies components that are most used in the application

<table>
<thead>
<tr>
<th>New platform knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>- often struggle with some of that stuff and sometimes just compromises; - New ideas for an app can come from new functionality in a new version of the platform; - Looking at different platforms for ideas; those requirements sessions that really put together the vision and how we were going to develop it; - always evolving; - The biggest challenge was getting that M player which is -- I believe that now, in some of the latest versions of iOS completely deprecated; it's gone; - constantly coming up with new devices, new functionality and the functionality; - deprecate the old players; - a skill set we've build over the years of developing to be able to take these new tools and apply them to a business; - New components introduced by the platform; deprecated components;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Converging Cycle of Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converging technological (means) constraints</td>
</tr>
<tr>
<td>- very few limitations as far as mobile devices go; - weigh the complexity, what tools are required, how many resources, are there any dependencies; - little bit innovative; - the user delegate system was challenging; - Microsoft Azure’s SQL database; client wanted HTML5 player but was not possible;</td>
</tr>
<tr>
<td>Converging feature constraints</td>
</tr>
<tr>
<td>- Iterations; - time and financials are our biggest constraints when it comes to mobile development;</td>
</tr>
</tbody>
</table>
- We were all working together to compromise on something;
- there was definitely a lot of requirements and features that were taken out because of either value-add or budget;
- expectations were very high; they make tweaks;
- I appreciate you wanting to and you want him to make this solution better but that's not what we've estimated and that's not what the budget allows for and that's not what we agreed to in the sprint;
- usually got to something where they were happy, whether we had to change directions;
- list of features that we want to build;
- Backlog grooming

<table>
<thead>
<tr>
<th>Converging platform constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Limits and restriction;</td>
</tr>
<tr>
<td>• Apple has to keep deprecating support for their older phones when they introduce new iOS, because the power is not there anymore as they want to add new features;</td>
</tr>
<tr>
<td>• work with our customers very well to understand that the amount of time we have versus their expectations and to help them manage that appropriately;</td>
</tr>
<tr>
<td>• with iOS, it's always a moving target; The platform itself has limitations?</td>
</tr>
<tr>
<td>• Developer: Yes. The control itself that we were using in iOS has limitations so we had to extend it to support.</td>
</tr>
<tr>
<td>• “Luckily we had a guy on our team who wrote a book on video playback on iOS so he was really really good and he was able to do that for us;”</td>
</tr>
<tr>
<td>• Xamarin didn't support as many iOS like everything that iOS supported out of the box;</td>
</tr>
<tr>
<td>• new broadband networks that Android takes advantage of, that iOS does not yet; M-player versus HTML player</td>
</tr>
</tbody>
</table>
### APPENDIX 7  TB CASE STUDY FINDINGS

#### Table 15. TB Case Findings

<table>
<thead>
<tr>
<th>Construct</th>
<th>First Cycle Code</th>
<th>Case Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Means (existing resources at hand)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>• Development Process: prioritize backlog list, and when there is “a good chunk that we feel are the next priority in our queue” create a 4 week sprint, track with Jira, “assign tasks to a user stories, and task to a sprint”. then they are assigned to architects, or to the application’s product owner, or to consultants; content for future development; • How development proceeds depends on how big the idea is; • Have to decide whether a customer request should be added to the product (preferred) or a one-off development for that customer; • Uncertainty about some data volume and sources; • Technical challenge from functions that are what the user wants and easy to use, but may reduce performance or not sustainable; • Interoperability with other applications; Future additions; • standards on our APIs and portals; various modules; • using a lot of latest cutting edge technologies; • I wouldn't say we're 100% Agile by any means; Microsoft Azure and Dynamic 365; Dot Net framework; sandbox environments;</td>
</tr>
<tr>
<td>Market knowledge</td>
<td></td>
<td>• Different Sources of knowledge – existing and prospective customers, regulations; • Customers are the best sources; • Context-specific impact on development: When there is a new suggestion from a user, it is vetted because often there is a tradeoff between efficiency and quality of patient care; • vetting with that type of very visual; • Technical team input; Sources of information for product development; • Another part that sets us apart is our content; Original idea for application: “the original idea goes back to say, &quot;Well, we're going to have a perpetual care plan that</td>
</tr>
</tbody>
</table>

172
manages a holistic care of this patient across all care team members." Goes beyond another application – knew coordination of care would be needed for ACA;

- Future needs; Things change rapidly in this market; Partners; existing customers; industry reports; regulations

<table>
<thead>
<tr>
<th>Platform knowledge</th>
<th>Leverage the platform; Platform requires changes; restrictions or certain quirks with the way the platform is set up; Decided not to add a new requirement because of the platform; Sources of platform knowledge – technical team, webinars, internal knowledge transfer; biggest differentiator that stands out versus we is the only healthcare solution out there based upon the serial platform; Microsoft platform is, it's pretty open; provides a set of standard sales service and marketing capabilities that are not specific to any one industry; some data manipulation, some Power BI embedding; Keeping up with platform changes – conferences; Changes in platform; Problems with platform – different names for same thing; users' perspective the platform is almost-- it's becoming meaningless. As you talk cloud, they don't know what cloud it is, they don't care what cloud it is; Need to change platform upon which the application is built; Release notifications;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Social Capital</td>
<td>Collaborative process within the development team when not in a sprint; Collaborative process within the development team when in a sprint; Internal knowledge transfer; feedback; to review the product as a team; Informal ways to keep up with the platform; Key team member for understanding the market and product and act as a liaison; Content partnerships, as with name of university; Learning culture at TB; Mix of market, product and platform knowledge; Leverage new employees; Sharing culture; Visuals help bridge technical-user gap;</td>
</tr>
<tr>
<td>Platform (resources and constraints provided by the platform)</td>
<td>Value</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Technology (API)</td>
<td>Market</td>
</tr>
<tr>
<td>Culture</td>
<td></td>
</tr>
<tr>
<td>• Informal review of the product; Conferences</td>
<td>• Frequent meetings</td>
</tr>
<tr>
<td>• Informal review of the product; Conferences</td>
<td>• Open communication channels</td>
</tr>
<tr>
<td>• Informal review of the product; Conferences</td>
<td>• Individual autonomy to trial ideas</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• flexibility and the usability in configuring things is a real bonus of the Microsoft platform;</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• costs of obtaining the various Microsoft platform products;</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• Platform based development speeds deployment;</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• limited UI and functions because of platform;</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• “the benefits or limitations of the Microsoft as your platform; Artificial intelligence; reporting; database; user interface; integration capabilities</td>
</tr>
<tr>
<td>• Frequent meetings</td>
<td>• licensing costs for the Microsoft platform</td>
</tr>
<tr>
<td>• Open communication channels</td>
<td>• Platform adds value; Platform helps integration with other applications; easy to learn;</td>
</tr>
<tr>
<td>• Open communication channels</td>
<td>• Different platforms in use have different capabilities and so choice between them in development; Platform change;</td>
</tr>
<tr>
<td>• Open communication channels</td>
<td>• House metaphor for advantage of platform-based development: “I think it definitely advanced how far and how quick the development team has gotten with the solution. We started with a very strong base, it’s almost like you’re buying the house with all the framing up and all you need to do is to bring in the furniture and the decoration. I think it really has probably expedited the development by building on Microsoft.”;</td>
</tr>
<tr>
<td></td>
<td>• Leveraged and extended the platform;</td>
</tr>
<tr>
<td>• Individual autonomy to trial ideas</td>
<td>• Choice between platforms for certain tasks; familiar to use; biggest differentiator that stands out versus we is the only healthcare solution out there based upon the serial platform;</td>
</tr>
<tr>
<td>• Individual autonomy to trial ideas</td>
<td>• Advantage of platform for scaling and flexibility; flexible to add or modify;</td>
</tr>
<tr>
<td></td>
<td>• Value of feature determines feature selection; Microsoft platform is, it's pretty open;</td>
</tr>
<tr>
<td></td>
<td>• Integration via platform to other applications;</td>
</tr>
<tr>
<td></td>
<td>• restrictions or certain quirks with the way the platform is set up;</td>
</tr>
<tr>
<td></td>
<td>• Integration to other applications;</td>
</tr>
<tr>
<td>Maturity</td>
<td>• Platform is mature</td>
</tr>
<tr>
<td></td>
<td>• Scheduled releases</td>
</tr>
<tr>
<td></td>
<td>• Pre-release communications</td>
</tr>
</tbody>
</table>
| Complexity | • Platform is complex due to number of components  
|           | • Introduction of new components with additional functionality |
| Aspirations | • Ease of use, especially in ease of navigation is key value of product, as well as patient-centered view and interoperability;  
| Product-market match | • Tension between product for a specific customer and general market; “Healthcare has a lot of standards for compatibility around data exchange. We're very cognizant of those.”;  
| | • Backlog facilitates continuous improvement;  
| | • Enhancements during testing;  
| | • Success criteria would be having satisfied customers, and having ongoing sales speeds, and continually having new customers sign on; functional team comes up with the dream, "How do we want it work ideally.";  
| | • Informal idea generation;  
| | • Constant change is the struggle;  
| | • Requirements determination - try and listen to the customers and then their demand;  
| | • After implementation review for enhancements;  
| | • Build and maintain expectations for the product throughout;  
| | • Sales and developers are integrated;  
| | • Fit with what the market wants in general vs. a specific customer;  
| | • The market niche for this product is outside of any one healthcare provider – but focused on the patient both at home and interacting with any and all providers – “What the application does is help bridge that gap that most providers have.”;  
| | • Market demand for more data;  
| | • Product-market match is most important;  
| | • Balance between what the market knows it wants and what could be done;  
| | • Visual interface helps with ease of use; market trends;  
| | • Application addresses requirements of users |
| Product-platform match | • Tradeoff between what possible between visual design and technical implementation;  
| | • Our decision is always driven by whether it will work on this Microsoft platform;  
| | • Platform does not support some feature;  
| | • Mix of platforms helps; |
| **Exceed Platform Value** | - match upgrades made in Dynamics 365 or to start using Azure or various things related to our platform;
- Multiple platforms help with breadth of application; more flexible and scalable and much more robust operates; flexible to add or modify;
- all five modules be on one platform;
- Need to integrate with input from and export to multiple applications;
- UI of it sometimes is limited because of the Microsoft platform;
- decision is always driven by whether it will work on this Microsoft platform;
- Architect would like to branch out to using other platforms;
- users' perspective the platform is almost-- it's becoming meaningless;
- Application uses platform’s components

| **Novelty** | - “We are trying to exceed the Dynamics 365 value by making it tailored to healthcare because there are not a lot of other products that do that.”; the ultimate dreamers;
- leverage what we can from the platform while also optimizing;
- Enhancements beyond the platform;
- Application extends platform’s capabilities in Healthcare setting

| **Novelty** | - integrated with all of Microsoft products;
- biggest differentiator that stands out versus we is the only healthcare solution out there based upon the serial platform;
- layer our IP on top of that we completely exceed the base platform features;
- predicting what could happen based on the data that we already have;
- A multifunction and mature product that is “much more flexible and scalable and much more robust.”;
- “multi-morbidity care plan stacking” is “a major differentiator”;
- Ease of use, patient-centered data model (a more holistic approach), and “interoperability in terms of open standard interfaces” for data import and export;
- changed the data model; “I haven't seen any competitors that are purely on the Azure platforms;
- “Another part that sets us apart is our content; integrated with the Microsoft platform, and all of Microsoft products; definitely help with the marketability”;
- “our value and our novelty is weighted more highly certainly than our product platform match.”;
- one place for everyone to manage and coordinate the care of that patient no matter how many conditions they have, no matter how many health care providers they are seeing, it's one care plan;
- using a lot of latest cutting edge technologies;
- adopt a cloud and embrace the cloud services as holistically;
- Competitors can imitate because on a platform;
- Novel within TB;
- Application provides an integrated patient focused management system

<table>
<thead>
<tr>
<th>Acceptable Risk</th>
<th>Logic of Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit limited resource</td>
<td>Logic of Control</td>
</tr>
<tr>
<td>Application recoverable after failure</td>
<td>Logic of Control</td>
</tr>
<tr>
<td>Risk Analysis</td>
<td>Logic of Control</td>
</tr>
</tbody>
</table>

- Deliberations on when to commit resources;
- Conflict that arises because of limited resources;
- Stories;
- spend limited time for proof of concepts

- rigorous about our testing; make sure we did this right.”

- biggest risk I think is disruption;
- risk of breaking something;
- identifying risks;
- aspirations for value and novelty drive the acceptable risk probably more than this [the model] suggests.”;
- acceptable because we had had other platforms at our disposal;
- Risk from impending platform updates; you need to try it knowing that there's risk;
- Consider risk of implementing feature

- Impact of platform upgrade that is impending; Microsoft round;
- Decision making around development of IP for the product versus one-off development for a particular client;
- we decide what's going to a release;
- Decision making on which tasks to undertake depends on multiple factors;
- Decision making around new platform release;
- Iterative decision making during feature development;
- Decision making in light of new platform releases;
- Sales team decision making; Approval to work on new features; Decisions depend on size of new potential feature; prioritize what features and functions we build;
Sometimes prospective customers ask for a feature, but existing customers vet it and can determine if it is a good idea or not;

- After implementation of a one-off feature it is reviewed to see if it should be added to the product for everyone—although the customer may not want this for competitive reasons;
- If a new release of the platform enables a feature that was determined to be too costly or hard in the past, it is reconsidered for development;
- Product owner has key role; sprint short enough; “I don't think we've had a lot of cases where we've had user stories that match potential product capability, but not current capability.”;
- Technical architects advise on which aspirations can be pursued

<table>
<thead>
<tr>
<th>Action</th>
<th>Fixed bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed Tasks</td>
<td>Development servers;</td>
</tr>
<tr>
<td></td>
<td>quickly test while it's still in development</td>
</tr>
</tbody>
</table>

### Effects

| NA | User stories, visual designs, a functional/technical aspect delivered to the developers, explanations of designs, documentation, CRM workloads, design and create the forms and testing, walkthroughs of screenshots, mockups, roadmaps of future feature development plans, prototypes, feedback from customers, daily stand call, project kickoffs, estimates of effort and cost for development; |
| | Communication about changes; |
| | Data migration and integration of important and sometimes challenging activities—often there are unanticipated problems with bringing in legacy data; |
| | Deliverables at 4-week sprints |

<table>
<thead>
<tr>
<th>Expanding Cycle of Resources</th>
<th>New technological knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process of enhancement from technical developers;</td>
</tr>
<tr>
<td></td>
<td>Constant need for the technology staff to keep up with rapidly changing tools and constantly learn;</td>
</tr>
<tr>
<td></td>
<td>New technical possibilities with new releases of the platform;</td>
</tr>
</tbody>
</table>
| New market knowledge | • always trying to improve our products so we have a backlog list of user stories or enhancements;  
• partnered with a few different vendors to provide content;  
• prospective customers or existing customers suggest modification;  
• functional team comes up with the dream;  
• Process for considering new additions;  
• Consider enhancements after implementation of a release;  
• Functionality in current release is the starting point; Partner delivered us content; partnerships;  
• Crucial to have enough customers in all subparts of the application;  
• Newly hired employees can offer a fresh perspective;  
• Data collected about use can help with understand what is needed;  
• Decision about release features;  
• planning: A lot of the new features, either are already on our road map, our people will talk about what we're going to accomplish this year and how we need to extend the product or what pieces look important.  
• We also listen to our customers. Our customers are probably some of the best people for identifying new features or because they're going to ask once they get used to that now--Now they're going to ask for more stuff. How to extend it and how to make it better.  
• When they are using our tools, what's next.”; try not to react on anything; During development sharing helps build knowledge of the product; New requirements; intellectual property; customization |
|---|---|
| New platform knowledge | • Understanding the impact of the platform both enabling and constraining; platform provide new ideas to your team;  
• latest Microsoft cognitive services APIs;  
• tools are getting easier to use, quicker to use and they're providing more functionality across the table;  
• New releases from platform |
| Converging Cycle of Constraints | Converging technological (means) constraints | • the functional team comes up with the dream;  
• cone of uncertainty – “There's always something that you uncover. That would be one thing, the other things we've come up against is a customer, they may have a legacy system that we've never worked with before and they insist they can get the data out. Then it turns out that no, they can’t get the data out or maybe they don't actually
| Converging feature constraints | • Changes can create need for another change; Informal enhancements;  
|                               | • Feature removal; “I feel like the struggle is that our priorities are always changing, and our requirements are always changing.”;  
|                               | • degrees of identifying a gap in the product but typically it gets us that 80-90%;  
|                               | • sales pursuits generate new in-markets requirements;  
|                               | • Springboard customers; internal vetting process;  
|                               | • wish list; “Our decision is always driven by whether it will work on this Microsoft platform; rigorous about our testing; outdated pieces are retired; non-relevant stuff, to a great extent probably remains just because it's harder to-- you never know when it's going to be relevant again, first of all, so you tend not to take things away. You may turn them off but they're still there. It's more about the new things and making it all fit, making it all look right; Ideas arise during implementation;  
|                               | • Deciding about new tasks to assign;  
|                               | • dependencies to old entities;  
|                               | • Intellectual property versus customization |

| Converging platform constraints | • very rare that we come across something that we just can't do; “sometimes Microsoft can be limiting but I don't blame them. I mean it is their ecosystem;  

own the data. The legacy system vendor is not going to play nice and those aren't as often but it still happens.”;  
• Choice of APIs, framework
APPENDIX 8  INSTITUTIONAL REVIEW BOARD APPROVAL
July 14, 2017

Onkar Malgonde
Information Systems & Decision Sciences
CIS 2055
4202 E. Fowler Avenue
Tampa, FL 33620

RE: Expedited Approval for Initial Review
IRB#: Pro00031044
Title: An Effectual Approach for the development of Novel Applications on Digital Platforms

Study Approval Period: 7/14/2017 to 7/14/2018

Dear Mr. Malgonde:

On 7/14/2017, the Institutional Review Board (IRB) reviewed and APPROVED the above application and all documents contained within, including those outlined below.

Approved Item(s):
Protocol Document(s):
Protocol Version #1

Note, no research activities can begin without submitting the required letter of support and receiving an approval through the Amendment process.

Consent/Assent Document(s)*:
Informed Consent V1.pdf.pdf

*Please use only the official IRB stamped informed consent/assent document(s) found under the "Attachments" tab. Please note, these consent/assent documents are valid until the consent document is amended and approved.

It was the determination of the IRB that your study qualified for expedited review which includes activities that (1) present no more than minimal risk to human subjects, and (2) involve
only procedures listed in one or more of the categories outlined below. The IRB may review research through the expedited review procedure authorized by 45CFR46.110. The research proposed in this study is categorized under the following expedited review category:

(6) Collection of data from voice, video, digital, or image recordings made for research purposes.

(7) Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.

As the principal investigator of this study, it is your responsibility to conduct this study in accordance with IRB policies and procedures and as approved by the IRB. Any changes to the approved research must be submitted to the IRB for review and approval via an amendment. Additionally, all unanticipated problems must be reported to the USF IRB within five (5) calendar days.

We appreciate your dedication to the ethical conduct of human subject research at the University of South Florida and your continued commitment to human research protections. If you have any questions regarding this matter, please call 813-974-5638.

Sincerely,

Kristen Salomon, Ph.D., Vice Chairperson
USF Institutional Review Board