Running head: EFFECTS OF AGILE METHODS ON WEBSITE QUALITY FOR E-COMMERCE

EFFECTS OF AGILE METHODS ON WEBSITE QUALITY FOR ELECTRONIC COMMERCE

David F. Rico

(August 29, 2007)

Dissertation submitted to the faculty of the
Graduate School of Management and Technology
University of Maryland University College

in partial fulfillment of the requirements for the degree of

DOCTOR OF MANAGEMENT

(Information Technology)

Doctoral Committee:

Hasan H. Sayani, PhD, Chair
James J. Stewart, PhD, Member
Ralph F. Field, PhD, Member
Jeffrey V. Sutherland, PhD, Member

ABSTRACT

The purpose of this dissertation is to study the relationships between the use of agile methods to manage the development of Internet websites and website quality. An agile method is a new product development process that is often associated with the Internet software industry. Agile methods are characterized by factors of iterative development, customer feedback, well-structured teams, and flexibility. Use of agile methods may improve software quality by injecting customer feedback into a stream of working software versions to converge on a solution. Surveys of software professionals were used to help determine whether the use of agile methods improves the quality of Internet websites for the $2 trillion U.S. electronic commerce industry. The central contributions of this study include a literature review, conceptual model, survey instruments, measurement data, and data analysis for agile methods and Internet website quality. Additional contributions include a thorough discussion of the relationships between agile methods and website quality, the results of which may help U.S. managers better understand the effects agile methods. Our findings indicated iterative development and customer feedback were related to website quality, but well-structured teams and flexibility were not. Finally, we included a list of recommendations for researchers and scholars who may wish to study these relationships further.

Keywords. Agile methods, iterative development, customer feedback, well-structured teams, flexibility, website quality.

EXECUTIVE SUMMARY

Electronic commerce has grown into a $2 trillion U.S. industry that involves the buying and selling of products and services utilizing Internet websites such as Amazon or E-bay. As many as 67% of U.S. firms use agile methods to develop their Internet websites, which is a new product development process that emphasizes iterative development, customer feedback, well-structured teams, and flexibility to improve website quality. Do agile methods improve website quality? Do agile methods based on job-shop or craft-industry principles improve website quality better than traditional ones from the scientific management era?

Survey data was collected from 250 respondents to determine if the use of iterative development, customer feedback, well-structured teams, and flexibility improved the quality of their websites. The components of this study include: (a) a history of electronic commerce and agile methods, (b) a conceptual framework, (c) survey instruments, (d) a repository of original data, and (e) a roadmap for conducting future studies of agile methods. Analysis showed that iterative development and customer feedback improved website quality, but well-structured teams and flexibility did not. Some of the more notable findings include:

- 70% of all developers are using many if not all aspects of agile methods.
- 79% of all developers using agile methods have more than 10 years of experience.
- 83% of all developers using agile methods are from small to medium-sized firms.
- 26% of all developers using agile methods have had improvements of 50% or greater.
- Developers using all aspects of agile methods produced better e-commerce websites.

The findings also showed that iterative development is only effective if all iterations are working software. The lag time for getting customer feedback continues to plague software developers. Larger organizations may be shunning the use of agile methods to build bigger and more complex systems. Developers may not always be exploiting the most flexible technologies to their advantage. And, agile methods are absent at the edges of the adoption curve. In summary:

- 46% of organizations still struggle to produce operational code for all iterations.
- 70% of organizations don’t get timely, high-quality feedback from their customers.
- 85% of organizations are hesitant to use agile methods to develop larger systems.
- 57% of organizations are not using agile and flexible information technologies.
- 87% of programmers with less than 6 years of experience are not using agile methods.

The results of this study should be interpreted with caution since less than half the people who claim to be using agile methods may not be producing working software at the end of their iterations. This is contrary to a major principle of agile methods, which requires working software at the end of iterations. Some may argue that organizations that are not producing working software at the end of their iterations are not using agile methods. This observation has a two-fold impact on the results of this study. First, organizations who want to use agile methods should make a concerted effort to produce working software at the end of their iterations. Second, organizations may not actually be using agile methods if they are not producing working software at the end of their iterations. Again, some may argue that all principles of agile methods must be used in order to get all of the potential benefits of agile methods such as website quality.

PREFACE

Does use of agile methods improve the quality of Internet websites used for the $2 trillion electronic commerce industry by U.S. firms? Agile methods are a new product development process consisting of iterative development, customer feedback, well-structured teams, and flexibility. Use of agile methods may improve website quality by injecting customer feedback into a stream of working software versions to converge on a solution.

There is a well-documented debate among management scholars about the best approaches for managing the development of Internet websites. Some scholars believe traditional methods rooted in well-established scientific management principles lead to high quality Internet websites. However, other management scholars believe agile methods have the characteristics of a job-shop or craft industry predating the scientific management era and are the best approach for managing the development of high-quality Internet websites.

Our challenge is to identify, survey, select, or develop a scholarly instrument for measuring the use of agile methods and the quality of Internet websites, and then collect data to determine whether their use is linked to higher quality websites for electronic commerce by U.S. firms. Though agile methods are a general purpose approach for managing the development of new products, they are often associated with Internet software, which is the focus of this study.

There seems to be no middle ground on this issue. Some management scholars firmly believe agile methods lead to lower quality websites and others believe agile methods lead to higher quality websites, with neither side offering much empirical evidence to support their claims. We hope to create the first in a long line of scholarly studies that test the relationships between the use of agile methods as a management approach for developing Internet websites and improved website quality for electronic commerce among U.S. firms.

Internet websites are the fundamental tool for conducting business transactions on the Internet, otherwise known as electronic commerce. The U.S. alone generates around $1.95 trillion in revenues using Internet websites each year. Likewise, U.S. firms commit between $152 and $231 billion in information technology expenditures each year to achieve these revenues. Currently at stake is a ten-fold return on investment in electronic commerce revenues to information technology expenditures by U.S. firms each year (e.g., revenues ÷ expenditures).

There are more than 250,000 information technology projects in the U.S. each year and up to two-thirds of these projects have failed or are failing. Some argue that principles of agile methods such as iterative development, customer feedback, well-structured teams, and flexibility may improve project success. To that end, U.S. managers have already committed between 50% and 67% of their projects to the use of agile management methods. They may want to know whether the use of agile methods for managing the development of Internet websites leads to higher quality websites and what the potential consequences to lower electronic commerce revenues are if the detractors of agile methods are indeed correct—that is, whether traditional methods rooted in the scientific management era are superior to those of agile methods from the much earlier job-shop or craft industry of the early industrial revolution in the U.S. The lessons associated with both sides of these debates will be addressed throughout this study.

DEDICATION

This dissertation is dedicated to my wife, Celia, who patiently worked so hard over the last four years in order to make this degree possible. This dissertation is also dedicated to my children, David, Serena, Faith, and Christian, who sacrificed so many of their formative years for the fleeting hope of a brighter future for all of us. And, of course, to my family, friends, and colleagues who diligently provided daily encouragement to me by eagerly listening to my research interests, in order to keep me challenged, focused, and intrigued with my dissertation topic for so long.

ACKNOWLEDGMENTS

I would like to thank my dissertation committee, especially my chair, Dr. Hasan H. Sayani, who was one of the best leaders I have ever had the pleasure of serving. And, of course, to my committee members Dr. James J. Stewart and Dr. Ralph F. Field who seemed to have a knack for knowing when to lend a helping hand. Special thanks also go to my external committee member, Dr. Jeffrey V. Sutherland, who took an interest in my topic and went above and beyond the call of duty to help me fulfill some of my more challenging graduation requirements. My cohort also deserves honorable mention for being a great team. Oh yes, and thanks to Scott Ambler and Jon Erickson for helping me with my data collection, and thanks to Dr. Monica S. Bolesta for introducing me to the skills necessary to complete my data analysis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Does use of agile methods improve the quality of Internet websites used for the $2 trillion electronic commerce industry by U.S. firms? Agile methods are a new product development process consisting of iterative development, customer feedback, well-structured teams, and flexibility. Use of agile methods may improve website quality by injecting customer feedback into a stream of working software versions to converge on a solution. A survey will be conducted to test the hypothesis that the use of agile methods may be linked to website quality.

The use of agile methods for Internet software was a reaction to the rise of traditional software development methods, which were too cumbersome, expensive, rigid, and fraught with failure rates ranging from 65% to 87%. Downsizing was the norm and traditional methods were being used by large corporations in decline, rather than young, energetic firms on the rise. Millions of websites were created overnight by anyone with a computer and a modicum of curiosity. Agile methods marked the end of traditional methods in the minds of their creators.

Traditional methods for managing software development were created when the first commercial computers began emerging in the 1950s. Scientists and engineers began creating increasingly more powerful and complex computer systems and inordinately complex computer programs beyond the comprehension of a single human. These early computer programs had millions of components to perform the simplest of operations giving rise to traditional methods. Traditional methods consisted of formal project plans, well documented customer requirements, detailed engineering processes, hundreds of documents, and rigorous testing.

Agile methods emerged with a focus on iterative development, customer feedback, well-structured teams, and flexibility. Internet technologies such as HTML and Java were powerful new prototyping languages, enabling smaller teams to build bigger software products in less time. Because it could be built faster, customers could begin to see finished software sooner and provide earlier feedback and developers could rapidly refine their software. This gave rise to closed-loop, circular, highly-recursive, and tightly-knit processes for creating Internet software.

Purpose of Study

The purpose of this study is to determine whether the use of agile methods results in better website quality than software methods based on traditional management principles. Since both agile and traditional methods are currently in widespread use by U.S. firms, this study may help determine whether agile methods have any impact at all on website quality. It is not the purpose of this study to repeat scholarship comparing traditional and agile methods, analyzing its component parts, or analyzing gaps in its individual parts based on external factors. Rather, this study proposes to analyze its factors *in toto* and focus on little studied areas such as its basic claims that it improves website quality, especially for the field of electronic commerce. Higher quality websites may be a stepping stone to improved organizational and market performance.

Scope of Study

Therefore, the scope of this study will be limited to an analysis of using agile methods for managing the development of electronic commerce websites. Only the major factors of agile methods will be examined by this study. Furthermore, the scope of this study will be limited to an empirical analysis of the links between the factors of agile methods and scholarly models of website quality. There are many important factors within the computer science and software engineering fields and a single study could not possibly address all of them. Examples include operating systems, programming languages, artificial intelligence, software architecture, domain engineering, open source software, and many others. It is not the purpose of this study to minimize their importance, but determine the relevance of using agile methods by U.S. firms. This study will not examine whether agile methods are appropriate for large systems or whether agile methods result in more or less maintainable systems. Scholarly evidence is still emerging related to these issues, and it is not the purpose of this study to support or refute these claims.

Rationale and Justification

Today, U.S. firms generate more than $2 trillion each year from electronic commerce, and Internet websites are a big part of this massive revenue stream (U.S. Census Bureau, 2006). As a result, the top 500 U.S. firms spend more than $231 billion per year on Internet related technologies in order to exploit this potential revenue (U.S. Department of Commerce, 2006). Furthermore, much of the annual $400 billion U.S. defense budget is devoted to information technology as well (Fulghum & Wall, 2004), which may increase interest in the results of this study. There are more than 250,000 software projects in the U.S. of which more than 72% have failed or are failing (Standish Group, 2000). Therefore, executives and managers of U.S. firms may benefit from the knowledge that agile methods may be linked to better website quality. There has never been a greater need for scholarly studies of agile methods.

Relevance and Importance

With regard to the relevance and importance of this study, U.S. firms may also garner indirect economic benefits from using agile methods to develop Internet software. For instance, the stock market rewards firms with higher market valuations for publicly committing to the use of management approaches such as agile methods (Przasnyski & Tai, 1999). Furthermore, the stock market also rewards firms with higher market valuations for renovating their information technology infrastructures (Davis, Dehning, & Stratopoulos, 2003). What these studies mean is that a firm's investors may simply reward companies with higher market valuations for adopting the use of agile methods. Therefore, understanding the strategic implications of using agile methods for developing Internet software is relevant and important to managers engaged in business development as well. It is important to note that small U.S. firms elect to delist from the stock market due to expenses such as legal fees associated with public listings (O'Connor, 2005).

Significance and Interest

Agile methods may be significant and interesting to a number of stakeholders. These may include managers and developers of Internet software. Managers may want to use software development approaches well suited for Internet technologies. Developers may want to focus on creating the best possible Internet software without the overhead of using traditional methods. Management scientists, on the other hand, are responsible for creating both traditional as well as agile management approaches for software development. So this study may help them understand the dynamics of creating Internet software. Agile methods may also be used to develop Web 2.0 technologies, which offer an optimal blend of capabilities associated with both personal computer and Internet software (O'Reilly, 2006). So this study may help managers and technologists understand the dynamics of creating both current and future Internet software.

Organization and Outline

This document consists of eight sections: Introduction; Research Problem; Literature Review; Conceptual Framework; Research Method; Data Analysis; Discussion, Results, and Conclusions; and Recommendations for Future Research. This Introduction describes the context for using agile methods. The Research Problem describes the major issues and delineates the boundaries of this study. The Literature Review describes the history of software methods and the justification for this study. The Conceptual Framework describes a theory, factors, subfactors and hypotheses for using agile methods. The Research Method describes the use of survey research for studying agile methods. The Data Analysis describes the statistical analysis of the data from this study. The Discussion, Results, and Conclusions describe the results of the hypothesis testing and the findings from this study. The Recommendations for Future Research describe new research directions identified as a result of conducting this study of agile methods.

RESEARCH PROBLEM

The Internet is a powerful new communication medium for conducting free-market style business transactions involving the instant exchange of billions of dollars on a worldwide scale. Primarily due the Internet industry's low market entry requirements, the 21st century shifted the balance of power away from industrial age firms. This enabled Internet firms to monopolize market share and achieve unprecedented levels of profitability (Vise, 2005). Likewise, it presents large challenges for managing the development of Internet software. Some firms manage the development of Internet software using principles of flexibility and agility, while other firms use traditional methods rooted in the scientific management era.

The challenge is to investigate, examine, and determine whether the use of agile methods for managing the development of Internet software is linked to website quality. There is scant literature that investigates the linkages between the use of agile methods and project outcomes, such as organizational performance (Cusumano & Selby, 1995). There is even some literature that links the use of agile methods to higher website quality for the field of electronic commerce (MacCormack, 1998).

However, the major tenets, principles, and factors of agile methods had yet to fully evolve and emerge at the time of some of these writings. Few, if any, studies examine the effects of all four of the factors associated with agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. Furthermore, few studies link any of these factors associated with agile methods to outcomes, such as website quality for electronic commerce, in a scholarly manner. Therefore, this study proposes to analyze the effects of all four factors of agile methods and then empirically link these factors to scholarly models of website quality for electronic commerce.

Research Background

The history of such research is characterized by numerous attempts to link the use of software methods to software quality. Programmers, computer scientists, software engineers, and management scientists have been trying to solve problems associated with computer programming such as productivity, quality, customer satisfaction, and time-to-market for more than five decades. Flowcharts and stepwise refinement were used to help ease the process of creating complex computer programs in the 1960s (Dijkstra, 1969; Knuth, 1963). IBM created software inspections to increase quality and productivity by an order of magnitude in the 1970s (Fagan, 1976). IBM used iterative releases to develop the software for NASA's space shuttle in the 1980s (Madden & Rone, 1984). IBM also used customer feedback to produce 30 billion lines of application software and garner $14 billion in revenue in the 1980s (Sulack, Lindner, & Dietz, 1989). Hewlett Packard saved $350 million using a myriad of software methods in the 1990s (Grady, 1997). Motorola successfully produced an error free paging system at 25 times the normal productivity levels (Ferguson, Humphrey, Khajenoori, Macke, & Matvya, 1997). Electronic Brokering Services designed a 65,000 lines-of-code Java system using team processes that conducted $1 billion worth of online trades per day, without error, in record time (Goth, 2000). General Dynamics has even noted order of magnitude improvements in quality and productivity using the capability maturity model (Diaz & Sligo, 1997). Hewlett Packard also experienced 50% to 500% improvements in quality, productivity, cycle time, and return on investment by using domain engineering (Lim, 1998). Yet, all of these breakthroughs linking software methods to improved software quality are not without their skeptics (Sassenburg, 2002). This is the background that establishes the context for seeking empirical evidence linking agile methods to improved website quality for the $2 trillion U.S. electronic commerce industry.

Goals and Objectives

The research goals and objectives are to gather information that might determine whether a link exists between the use of agile methods for managing the development of Internet software and website quality for electronic commerce. Therefore, the research goals and objectives are to examine the empirical links between the theoretical factors of agile methods and scholarly constructs of website quality in the field of electronic commerce. Conversely, the research goals and objectives of this study are also to determine whether iterative development, customer feedback, well-structured teams, and flexibility are not linked to scholarly models of website quality for electronic commerce. Negative correlations between the factors of agile methods and website quality for electronic commerce are just as important as positive ones.

Research Questions

The basic research area to be explored is whether the use of agile methods by U.S. firms is more effective than traditional approaches based on scientific management principles. A closely related question is how do firms manage the development of Internet software for the $2 trillion U.S. electronic commerce industry? Another closely related question is what management approaches are linked to website quality for U.S. firms? Furthermore, what factors are motivating the use of new management approaches such as agile methods?

Specific questions must also include whether the major factors of agile methods are linked to website quality among U.S. firms. Is the use of iterative development linked to website quality among U.S. firms? Is the use of customer feedback linked to website quality among U.S. firms? Is the use of well-structured teams linked to website quality among U.S. firms? Is the use of flexibility linked to website quality among U.S. firms? Equally important is whether the answer to these questions is “no.”

Terms and Definitions

A software method is defined as “an approach to the analysis, design, construction, and implementation of information systems” (Hirschheim, Klein, & Lyytinen, 1996). Agile methods are defined as “the application of time-boxed iterative and evolutionary development, adaptive planning, evolutionary delivery, and other values and practices that encourage rapid and flexible response to change” (Larman, 2004). Iterative development is defined as “an approach to building software in which the overall lifecycle is composed of several iterations in sequence” (Larman, 2004). Customer feedback is defined as “the early identification of problems, effective screening of ideas, reduction of design changes in later stages of development, and better definition of global market and opportunities” (Lim, Sharkey, & Heinrichs, 2003). Well-structured teams are defined as “groups who are responsible for setting direction, establishing boundaries, assigning staff with certain talents to roles, and building a multi-tiered decision-making process in which managers have the responsibility and authority to make certain decisions” (Highsmith, 2002). Flexibility is defined as “the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed” (Institute of Electrical and Electronics Engineers, 1990). Website quality is defined as “the extent to which a website facilitates efficient and effective shopping, purchasing, and delivery of products and services” (Gounaris, Dimitriadis, & Stathakopoulos, 2005). A U.S. firm is defined as “a business registered with a national regulatory authority” (Caccese & Lim, 2005). The Internet is defined as “a network of millions of computers, a network of networks, or an internetwork” (Reid, 1997, p. xvii). A website is defined as “a collection of web pages for sharing of business information, maintaining business relationships, or conducting business transactions using the Internet” (Kalakota & Whinston, 1996).

Assumptions and Constraints

The first assumption is that agile methods are a measurable phenomenon. Another assumption is that agile methods will be relevant in the near future. A subtle assumption is that agile methods are linked to website quality and electronic commerce. An overriding assumption is that scholarship on agility is not already mature. There are many topics in the field of information systems, one study cannot address them all, and this study will not attempt to do so. Special purpose methods address specific types of non-functional requirements such as security, safety, or maintainability. This study will not attempt to address special kinds of non-functional requirements. For instance, the instrument for measuring website quality will be kept to a small set of items, with proven inter-item reliability. An instrument addressing every kind of non-functional requirement would have hundreds of items, and respondents simply will not have time for such a survey (which is beyond the scope of this study).

One may argue that unless the entire organization is committed to using agile methods the effects of individual engineers and teams on website quality would be difficult to measure. Furthermore, an organization may have multiple project teams who use agile methods to varying degrees of success, so the positive effects on website quality may not reach users or customers. An important assumption is that engineers have responsibility for the whole product and for interfacing directly with their customers using agile methods. In this scenario, the engineer gets a little market intelligence, develops some operational software, gets customer feedback, and then repeats the cycle until the desired quality levels with the website are achieved. This scenario would break down quickly if one inserted a software development team using agile methods into a large traditional bureaucratic organization and then expected them to converge on a solution without any customer or market feedback.

LITERATURE REVIEW

The purpose of this section is to present a literature review relevant to a study of using agile methods to manage the development of Internet websites and their subsequent quality. This study places website quality within the context of the $2 trillion U.S. e-commerce industry. Thus, this section provides a history of electronic computers, electronic commerce, software methods, software quality metrics, agile methods, and studies of agile methods. None of these histories are without controversy. For instance, some scholars begin the study of the electronic computer by mentioning the emergence of the Sumerian text, Hammurabi code, or the abacus. We, however, will align our history with the emergence of the modern electronic computer at the beginning of World War II. The history of electronic commerce also has poorly defined beginnings. Some studies of electronic commerce begin with the widespread use of the Internet in the early 1990s. However, electronic commerce cannot be appreciated without establishing a deeper context.

The origins of the software industry are also somewhat poorly documented. It started sometime in the 1960s, though it did not really take off until the 1980s. Software methods, however, are often associated with the rise of structured design in 1969, though they can firmly be traced back to the late 1950s. The first uses of agile methods are also not well defined. Many associate agile methods with the appearance of extreme programming in 1999, though agile methods began taking hold in the early 1990s, and most of its principles appeared before 1975. While research on agile methods began appearing in 1998, therein lies the issue. Few scholarly studies, if any, have been performed on agile methods; thus the basic purpose of this literature review is to establish the context to conduct scholarly research within the fields of agile methods and electronic commerce. Since agile methods may be linked to outcomes such as software quality, an in-depth analysis of literature on software quality metrics is also included.

History of Computers and Software

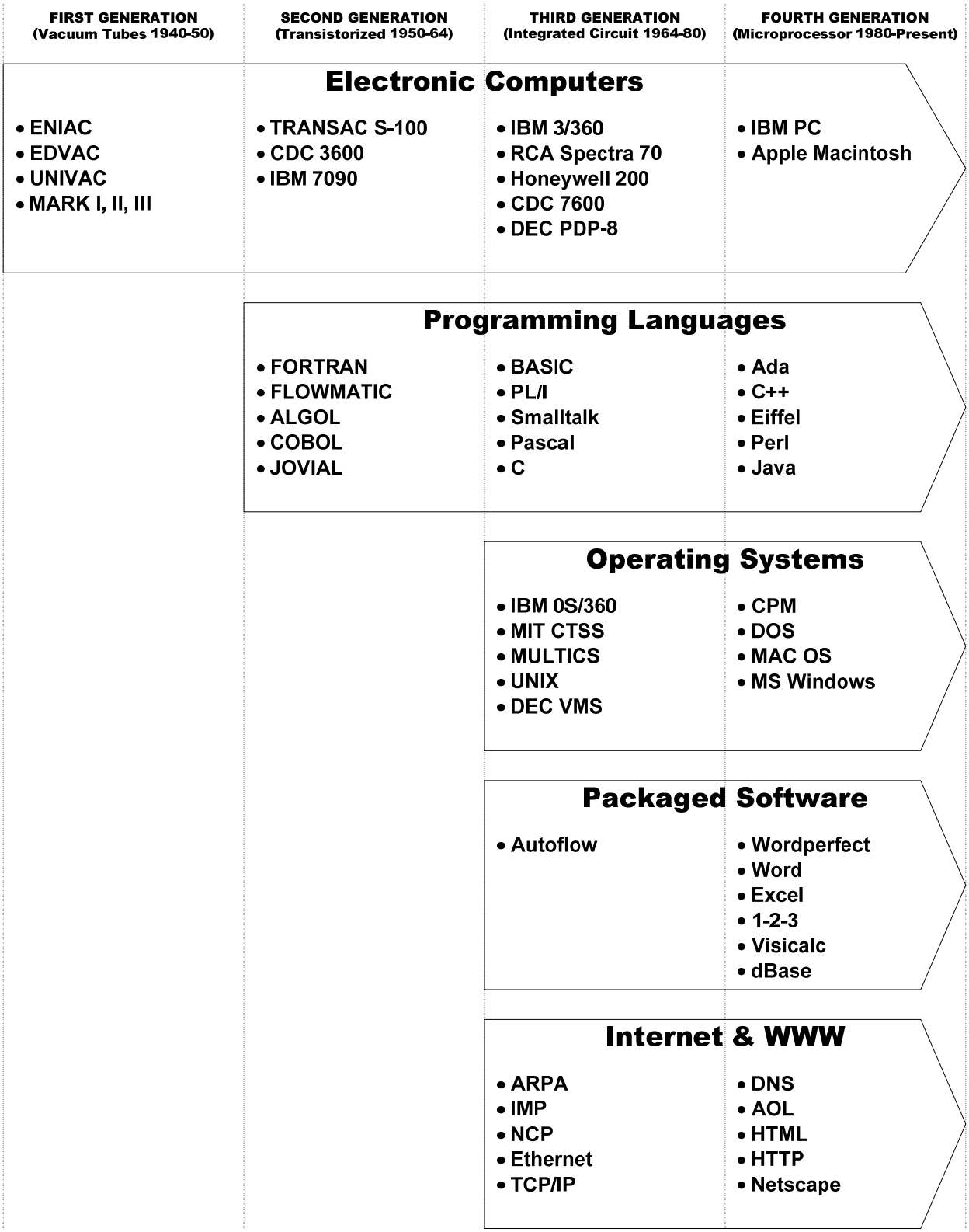

Figure 1. Timeline and history of computers and software.

Electronic computers. Electronic computers are simply machines that perform useful functions such as mathematical calculations or inputting, processing, and outputting data and information in meaningful forms (Rosen, 1969). Modern electronic computers are characterized by four major generations: first generation vacuum tube computers from 1940 to 1950, second generation transistorized computers from 1950 to 1964, third generation integrated circuit computers from 1964 to 1980, and fourth generation microprocessor computers from 1980 to the present (Denning, 1971; Rosen, 1969; Tanenbaum, 2001). First generation or vacuum tube computers consisted of the electronic numerical integrator and calculator or ENIAC, electronic discrete variable computer or EDVAC, universal automatic computer or UNIVAC, and Mark I, II, and III computers (Rosen, 1969). Second generation or transistorized computers consisted of Philco's TRANSAC S-1000, Control Data Corporation's 3600, and International Business Machine's 7090 (Rosen, 1969). Third generation or integrated circuit based computers consisted of International Business Machine's System/360, Radio Corporation of America's Spectra 70, and Honeywell's 200 (Rosen, 1969). Late third generation computers included Cray's CDC 7600 as well as Digital Equipment Corporation's PDP-8 and VAX 11/780 (Carlson, Burgess, & Miller, 1996). Fourth generation or microprocessor based computers included the International Business Machine's Personal Computer or PC and Apple's Macintosh (Tanenbaum, 2001).

Programming languages. Programming languages are defined as "any of various languages for expressing a set of detailed instructions for a digital computer" (Nerlove, 2004). By 1972, there were 170 programming languages in the U.S. alone (Sammet, 1972b) and today there are over 8,500 programming languages worldwide (Pigott, 2006). First generation or vacuum tube computers did not have any programming languages (Sammet, 1972b). Second generation or transistorized computers were characterized by an explosion of programming

languages, the most notable of which included formula translation or FORTRAN, flowchart automatic translator or FLOWMATIC, algorithmic language or ALGOL, common business oriented language or COBOL, Jules own version of the international algorithmic language or JOVIAL, and the list processing language or LISP (Sammet, 1972b). Third generation or integrated circuit based computers likewise experienced a rapid increase in programming languages, the most notable of which were the beginner's all purpose symbolic instructional code or BASIC, programming language one or PL/1, Smalltalk, Pascal, and C (Chen, Dios, Mili, Wu, & Wang, 2005). Fourth generation or microprocessor based computers continued the trend of introducing new programming languages, such as Ada, C++, Eiffel, Perl, Java, and C#.

Operating systems. Operating systems are simply a layer of software between the computer hardware and end user applications used for controlling hardware peripherals such as keyboards, displays, and printers (Denning, 1971). First generation or vacuum tube computers did not have any operating systems and "all programming was done in absolute machine language, often by wiring up plugboards" (Tanenbaum, 2001). Second generation or transistorized computers did not have any operating systems per se, but were programmed in assembly languages or using the early computer programming language called formula translation or FORTRAN (Tanenbaum, 2001). Third generation or integrated circuit based computers consisted of the first formalized multiprogramming operating systems and performed useful functions such as spooling and timesharing (Tanenbaum, 2001). Examples of third generation operating systems include IBM's Operating System/360; the Massachusetts Institute of Technology's compatible time sharing system or CTSS; the multiplexed information and computing service or MULTICS; the uniplexed information and computer system or UNICS, which became UNIX; and Digital Equipment Corporation's virtual memory system or VMS

(Tanenbaum, 2001). Fourth generation or microprocessor based computers consisted of the control program for microcomputers or CPM, disk operating system or DOS, Apple's Macintosh operating system or MAC OS, and Microsoft's Windows (Tanenbaum, 2001).

Packaged software. Software is defined as "instructions required to operate programmable computers, first introduced commercially during the 1950s" (Cusumano, 1991). The international software industry grew slowly in revenues for commercially shrink-wrapped software from about zero in 1964, to $2 billion per year in 1979, and $50 billion by 1990 (Campbell-Kelly, 1995; Steinmueller, 1996). It is important to note that the custom, non-commercially available software industry was already gaining billions of dollars in revenue by 1964 (Campbell-Kelly, 1995). First generation or vacuum tube computers did not have any software and "all programming was done in absolute machine language, often by wiring up plugboards" (Tanenbaum, 2001). Second generation or transistorized computers were characterized by bundled software, e.g., software shipped free with custom computer systems, and customized software such as International Business Machine's SABRE airline reservation system and the RAND Corporation's SAGE air defense system (Campbell-Kelly, 1995). Third generation or integrated circuit based computers saw the first commercialized shrink-wrapped software such as Applied Data Research's Autoflow flowcharting software (Johnson, 1998) and the total annual sales for commercial software were only $70 million in 1970 compared with over $1 billion for custom software (Campbell-Kelly, 1995). Following the U.S. Justice Department's antitrust lawsuit against IBM around 1969, commercial software applications reached over 175 packages for the insurance industry in 1972 and an estimated $2 billion in annual sales by 1980 (Campbell-Kelly, 1995). Fourth generation or microprocessor based computers represented the golden age of shrink-wrapped computer software and were

characterized by Microsoft's Word and Excel, WordPerfect's word processor, Lotus' 1-2-3 and Visicorp's Visicalc spreadsheets, and Ashton Tate's dBase database (Campbell-Kelly, 2001). By 1990, there were over 20,000 commercial shrink-wrapped software packages on the market (Middleton & Wardley, 1990). The international software industry grew to more than $90 billion for pre-packaged software and $330 billion for all software-related products and services by 2002 (U.S. Department of Commerce, 2003) and is projected to reach $10.7 billion for the software as a service or SAAS market by 2009 (Borck & Knorr, 2005).

Internet and WWW. The Internet is defined as a network of millions of computers, a network of networks, or an internetwork (Reid, 1997, p. xvii). First generation or vacuum tube computers were not known to have been networked. Likewise, second generation or transistorized computers were not known to have been networked together. Third generation or integrated circuit based computers gave rise to the Internet as we know it today. Third generation computers of the 1960s gave rise to packet switching theory, the first networked computers, the U.S. military's advanced research project's agency or ARPA, and the first interface message processor or IMP (Leiner et al., 1997; Mowery & Simcoe, 2002). Late third generation computers of the 1970s gave rise to the network control protocol or NCP, email, open architecture networking, Ethernet, transmission control protocol, Internet protocol and one of the first bulletin boards by Compuserve (Leiner et al., 1997; Mowery & Simcoe, 2002). Early fourth generation or microprocessor based computers gave rise to the domain name system or DNS and Prodigy and AOL were created (Leiner et al., 1997; Mowery & Simcoe, 2002). Middle fourth generation computers of the Internet era gave rise to the hypertext markup language or HTML, hypertext transfer protocol or HTTP, and Netscape, which caused the number of computers on the Internet to reach one million by 1992 and 110 million by 2001 (Mowery & Simcoe, 2002).

History of Electronic Commerce

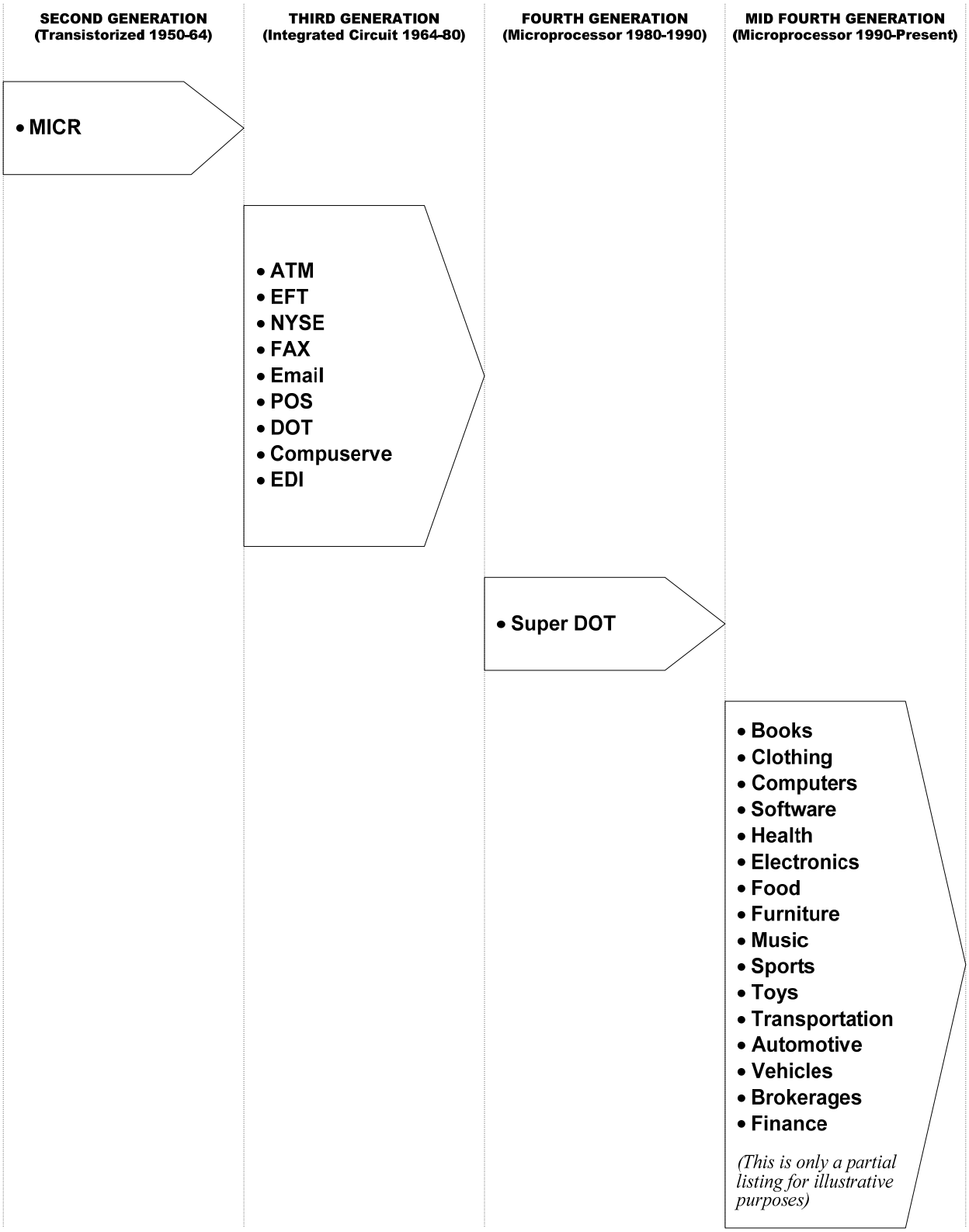

Figure 2. Timeline and history of electronic commerce.

Electronic commerce. From a simple perspective, electronic commerce is defined as sharing of business information, maintaining business relationships, or conducting business transactions using the Internet. However, there are at least four comprehensive definitions of electronic commerce (Kalakota & Whinston, 1996):

1. **Communications perspective**. Electronic commerce is the delivery of information, products, services, or payments via telephones or computer networks.
2. **Business process perspective**. Electronic commerce is the application of technology toward the automation of business transactions and workflows.
3. **Service perspective**. Electronic commerce is a tool to help firms, consumers, and managers cut service costs, improve quality, and speed delivery.
4. **Online perspective**. Electronic commerce provides the capability of buying and selling products and information on the Internet and other online services.

Electronic commerce is one of the most misunderstood information technologies (Kalakota & Whinston, 1996). For instance, there is a tendency to categorize electronic commerce in terms of two or three major types, such as electronic retailing or online shopping (U.S. Census Bureau, 2006). However, electronic commerce is as old as the computer and software industries themselves and predates the Internet era of the 1990s (Kalakota & Whinston, 1996). There is no standard taxonomy of electronic commerce technologies, but they do include major categories such as magnetic ink character recognition, automatic teller machines, electronic funds transfer, stock market automation, facsimiles, email, point of sale systems, Internet service providers, and electronic data interchange, as well as electronic retail trade and shopping websites (Kalakota & Whinston, 1996).

Second generation electronic commerce. Second generation or transistorized computers were associated with electronic commerce technologies such as magnetic ink character recognition or MICR created in 1956, which was a method of "encoding checks and enabling them to be sorted and processed automatically" (Mandell, 1977).

Third generation electronic commerce. Third generation or integrated circuit based computers were associated with electronic commerce technologies such as automatic teller machines, electronic funds transfer, stock market automation, facsimiles, email, point-of-sale systems, electronic bulletin boards, and electronic data interchange. In 1965, automated teller machines were created (Anonymous, 1965), which were electronic machines or computers that automatically dispense money or cash (Mandell, 1977). In 1966, electronic funds transfer or EFT was created (Ellis, 1967), which was "a set of processes that substitutes electronic messages for checks and other tangible payment mechanisms" (Mandell, 1977). Also in 1966, the New York Stock Exchange or NYSE was first automated (New York Stock Exchange, 2006). In 1971, facsimiles were created (Anonymous, 1971). In 1973, email was created (Mowery & Simcoe, 2002). In 1975, electronic point of sale systems were created (Anonymous, 1975), which involved "the collection in real-time at the point of sale, and storing in a computer file, of sales and other related data by means of a number of electronic devices" (Lynch, 1990). In 1976, the designated order turn around or DOT was created, which automated small-volume individual trades (New York Stock Exchange, 2006). In 1979, Compuserve launched one of the first electronic bulletin boards (Mowery & Simcoe, 2002). Also in 1979, electronic data interchange was created (Accredited Standards Committee, 2006), which is the "electronic movement of information, such as payments and invoices, between buyers and sellers" (Smith, 1988).

Fourth generation electronic commerce. Fourth generation or microprocessor based computers were associated with electronic commerce technologies such as the vast automation of the stock market. In 1984, the super designated order transfer 250 was launched to enable large scale automatic program trading (New York Stock Exchange, 2006).

Mid-fourth generation electronic commerce. Mid-fourth generation computers were associated with electronic commerce technologies such as selected electronic services, electronic retail trade, and electronic shopping and mail order houses. Selected electronic services consisted of industry sectors such as transportation and warehousing; information, finance, rental, and leasing services; professional, scientific, and technical services; administrative and support services; waste management and remediation services; health care and social assistance services; arts, entertainment, and recreation services; accommodation and food services; and other services (U.S. Census Bureau, 2006). Electronic retail trade consisted of industry sectors such as motor vehicles and parts dealers; furniture and home furnishings stores; electronics and appliance stores; building materials, garden equipment, and supplies stores; food and beverage stores; health and personal services; gasoline stations; clothing and accessories stores; sporting goods, hobby, book, and music stores; general merchandise stores; miscellaneous store retailers; and non-store retailers (U.S. Census Bureau, 2006). Electronic shopping and mail order houses consisted of industry sectors such as books and magazines; clothing and clothing accessories; computer hardware; computer software; drugs, health aids, and beauty aids; electronics and appliances; food, beer, and wine; furniture and home furnishings; music and videos; office equipment and supplies; sporting goods, toys, hobby goods, and games; other merchandise; and non-merchandise receipts (U.S. Census Bureau, 2006). Today, these are the only three recognized categories, garnering $1.95 trillion in 2004 (U.S. Census Bureau, 2006).

History of Software Methods

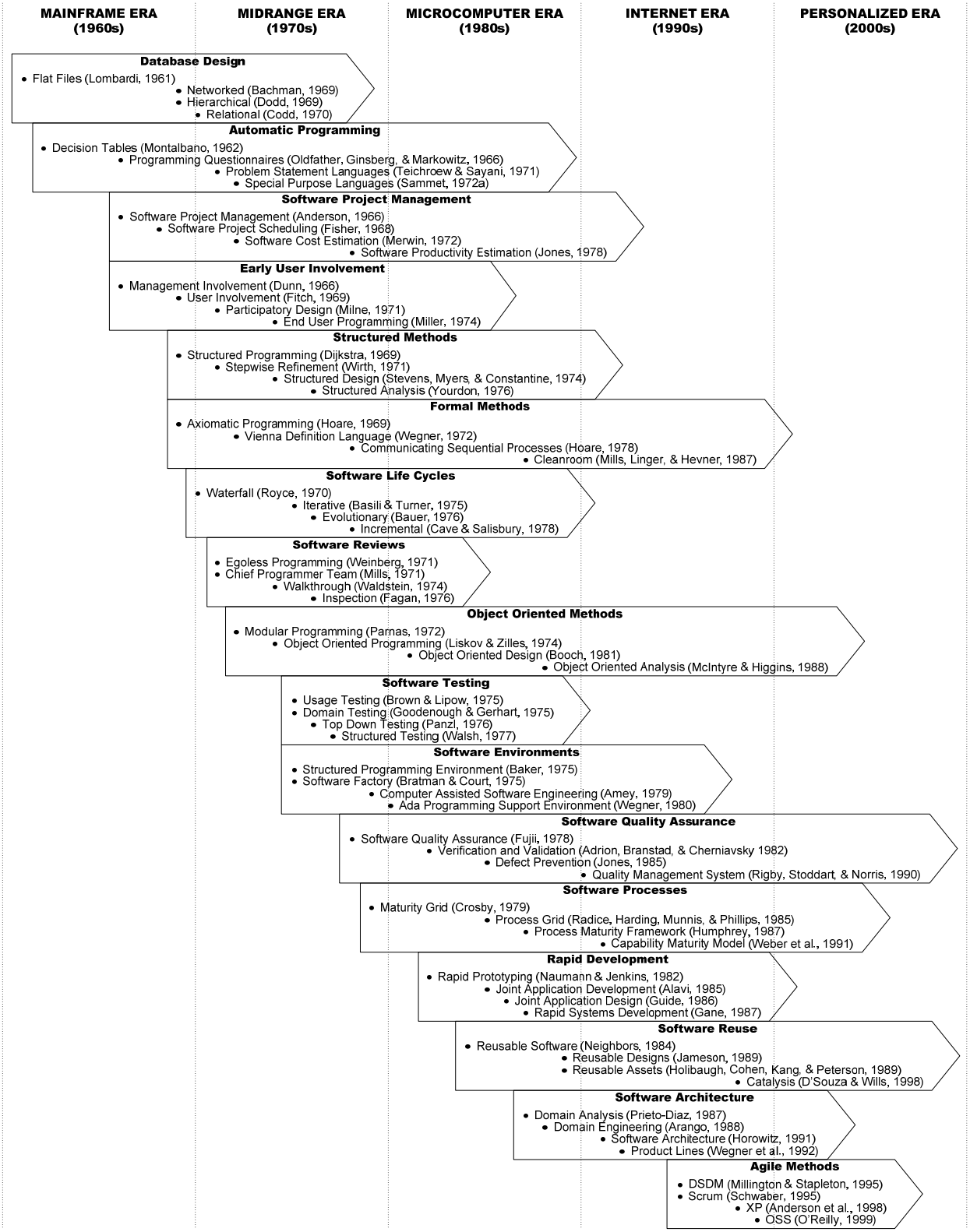

Figure 3. Timeline and history of software methods.

Database design. One of the earliest software methods emerging in the mainframe era of the 1960s was database design. Database design is a process of developing the structure and organization of an information repository (Teorey & Fry, 1980). The U.S. formed a standard information resource dictionary system or IRDS (American National Standards Institute, 1988), which is a "logically centralized repository of data about all relevant information resources within an organization, often referred to as metadata" (Dolk & Kirsch, 1987). The use of flat files for the design of information repositories was one of the earliest forms of database design (Lombardi, 1961). Network databases were one of the earliest forms of industrial strength information repositories consisting of many-to-many relationships between entities or data records, examples of which include IBM's Information Management System or IMS/360 and UNIVAC's Data Management System or DMS 1100 (Bachman, 1969). Hierarchical databases soon emerged with a focus on organizing data into tree like structures, which were believed to mimic the natural order of data in the real world (Dodd, 1969). Relational database design was introduced to create more reliable and less redundant information repositories based on the mathematical theory of sets (Codd, 1970).

Automatic programming. Another of the earliest software methods emerging in the mainframe era of the 1960s was automatic programming, which is also known as fourth generation programming languages or 4GLs. Automatic programming is defined as the "process and technology for obtaining an operational program and associated data structures automatically or semi-automatically, starting with only a high level user-oriented specification of the environment and the tasks to be performed by the computer" (Cardenas, 1977). Decision tables were one of the first automatic programming methods, which provided a simple format enabling both users and analysts to design computer software without being programmers themselves

(Montalbano, 1962). Programming questionnaires were also one of the first automatic programming methods, which provided an English-like yes or no questionnaire enabling non-programmers to answer a few discrete questions about their needs leading to the automatic production of computer software (Oldfather, Ginsberg, & Markowitz, 1966). The next evolution in automatic programming methods emerging in the early midrange era was problem statement languages, characterized by the information system design and optimization system or ISDOS, which provided a means for users and other non-programmers to specify their needs, requirements, and "what" to do, without specifying "how" the computer programs would perform their functions (Teichroew & Sayani, 1971). Special purpose languages also began emerging in the early midrange era. These languages were regarded as very high level, English-like computer programming languages used for rapid prototyping and quick software composition. Major examples include statistical analysis packages, mathematical programming packages, simplified database query languages, and report generators (Sammet, 1972a).

Software project management. The earliest notions of software project management also emerged in the mainframe era of the 1960s. An early definition of software project management consisted of the "skillful integration of software technology, economics, and human relations" (Boehm & Ross, 1988). The project evaluation and scheduling technique or PEST was one of the first complete approaches to software project management emerging in this era (Anderson, 1966). Project network diagrams in the form of the program evaluation review technique or PERT and the critical path method or CPM, though not originating in computer programming, were soon applied for planning, scheduling, and managing resources associated with software projects (Fisher, 1968). Cost estimation techniques were soon added to the repertoire of software project management, especially for managing large U.S. military projects

(Merwin, 1972). The framework for software project management was finally in place for years to come when basic measures of software productivity and quality were devised (Jones, 1978).

Early user involvement. Early user involvement has long been recognized as a critical success factor in software projects since the earliest days of the mainframe era. Early user involvement is defined as "participation in the system development process by representatives of the target user group" (Ives & Olson, 1984). While project overruns were considered a normal part of the early computer age, scholars began calling for management participation to stem project overruns which were now regarded as a "management information crisis" (Dunn, 1966). By the late 1960s, "user involvement" began to supplant management participation as a key to successfully designing software systems (Fitch, 1969). In the early 1970s, end users were asked to help design software systems themselves in what was known as "participatory design" (Milne, 1971). End user development quickly evolved from these concepts, which asked the end users themselves to develop the applications to help address the productivity paradox (Miller, 1974).

Structured methods. The late mainframe period gave rise to structured methods as some of the earliest principles of software engineering to help overcome the software crisis. Structured methods are approaches to functionally decomposing software designs, e.g., expressing software designs in high-level components, which are further refined in terms of lower-level components (Bechtolsheim, 1978). Structured programming emerged in this timeframe to help programmers create well-structured computer programs (Dijkstra, 1969). The next innovation in structured methods was called "top down stepwise refinement," which consisted of the hierarchical design and decomposition of computer programs (Wirth, 1971). Structured design, which is defined as "a set of proposed general program design considerations and techniques for making coding, debugging, and modification easier, faster, and less expensive

by reducing complexity," quickly followed suit (Stevens, Myers, & Constantine, 1974). Structured analysis methods rounded out this family of methods by suggesting the use of graphs for depicting the decomposition of software functions and requirements (Yourdon, 1976).

Formal methods. The late mainframe period also gave rise to formal methods, which would be used as the theoretical basis for software engineering for the next two decades. Formal methods are "mathematically based languages, techniques, and tools for specifying and verifying" reliable and complex software systems (Clarke & Wing, 1996). Axiomatic programming is one of the first recognized formal methods, which uses mathematical set theories to design functionally correct software (Hoare, 1969). An entire set of formal semantics was soon devised to serve as a basis for creating mathematically correct computer programming languages called the Vienna definition language (Wegner, 1972). The communicating sequential processes method was then created by the modern day founder of formal methods in computer science to help design mathematically correct multi-tasking software systems (Hoare, 1978). The cleanroom or box structured methodology was created to serve as a stepwise refinement and verification process for creating software designs (Mills, Linger, & Hevner, 1987). Formal methods, primarily due to the difficulty associated with their mathematical rigor, never enjoyed widespread adoption by the growing community of computer programmers (Shapiro, 1997).

Software life cycles. One of the first methods to come out of the early midrange era was the notion of software life cycles. A software life cycle is a "collection of tools, techniques, and methods, which provide roles and guidelines for ordering and controlling the actions and decisions of project participants" (Van Den Bosch et al., 1982). The waterfall is one of the first recognized software life cycles consisting of seven stages: system requirements, software requirements, analysis, program design, coding, testing, and operations (Royce, 1970). The

iterative software lifecycle appeared around the middle of the decade, which consisted of using a planned sequence of programming enhancements until computer software was complete (Basili & Turner, 1975). The evolutionary software life cycle soon formed with notions of gradually enhancing computer programs rather than developing them in phases or iterations (Bauer, 1976). The incremental software life cycle followed next, recommending project assessments at each major milestone in order to identify and reduce risk (Cave & Salisbury, 1978). The spiral model called for risk analysis between major milestones and prototypes as well (Boehm, 1986).

Software reviews. Software reviews emerged as a methodology in the very earliest stages of the midrange era. Software reviews are meetings held by groups of software engineers to review software products to identify and remove their defects (Sauer, Jeffrey, Land, & Yetton, 2000). Egoless programming was introduced in the early midrange era as a method of transforming software development from an individual craft into a loosely structured group activity (Weinberg, 1971). Chief programmer teams emerged shortly thereafter to formalize the notion of egoless programming with one small difference: the team would have a clear leader (Mills, 1971). Structured walkthroughs were quickly formed to once again place the responsibility for overall program quality in the hands of the team, rather than a single individual (Waldstein, 1974). Software inspections crystallized the concept of structured walkthroughs with a rigid meeting protocol for group reviews in order to optimize team performance (Fagan, 1976). That same year, the U.S. military formed a standard with system design reviews, software specification reviews, preliminary design reviews, critical design reviews, test readiness reviews, functional configuration audits, physical configuration audits, formal qualification reviews, and production readiness reviews (U.S. Department of Defense, 1976).

Object oriented methods. Object oriented methods emerged in the midrange era as a direct response to calls for a software engineering discipline to mitigate the software crisis. Object oriented methods are processes that "allow a designer to generate an initial design model in the context of the problem itself, rather than requiring a mapping onto traditional computer science constructs" (Rosson & Alpert, 1990). Modular programming was created with principles of information hiding, self contained data structures, co-located subroutines, and well-defined interfaces, which were claimed to improve efficiency, flexibility, and maintainability (Parnas, 1972). Object oriented programming was first elucidated in the middle of the midrange era, which demonstrated how the Simula programming language satisfied the principles of modular programming (Liskov & Zilles, 1974). Object oriented design emerged in the early microcomputer era to demonstrate one of the first graphical notations for describing object oriented programming languages (Booch, 1981). Finally, object oriented analysis methods emerged often reusing the tools of structured analysis to begin constructing specifications of software systems prior to devising their object oriented design (McIntyre & Higgins, 1988).

Software testing. Software testing gained recognition in the middle of the midrange era, though system and hardware component testing had been the norm for at least two decades. Software testing is defined as "the process of executing a software system to determine whether it matches its specification and executes in its intended environment" (Whittaker, 2000). Software usage testing was developed based on the notion that software reliability could be improved by specifying how computer programs will be used, devising tests to model how users operate programs, and then measuring the outcome of the testing (Brown & Lipow, 1975). Domain testing emerged at the same time with its principles of identifying test cases from program requirements, specifying a complete set of inputs using mathematical set theory, and

using set theory itself to prove program correctness when necessary (Goodenough & Gerhart, 1975). Soon thereafter, top down testing was introduced recommending a unique test procedure for each software subroutine (Panzl, 1976). Finally, structured testing emerged as an approach to encapsulate best practices in software testing for novice computer programmers (Walsh, 1977).

Software environments. Software environments emerged in the middle of the midrange era as a means of improving software quality and productivity through automation. A software environment is an "operating system environment and a collection of tools or subroutines" (Leblang & Chase, 1984). Structured programming environments were created as a means of improving software reliability and productivity using guidelines, code libraries, structured coding, top down development, chief programmer teams, standards, procedures, documentation, education, and metrics (Baker, 1975). Software factories were soon created to introduce discipline and repeatability, software visualization tools, the capture of customer needs or requirements, automated software testing, and software reuse (Bratman & Court, 1975). Computer assisted software engineering or CASE was also created to enhance software productivity and reliability by automating document production, diagram design, code compilation, software testing, configuration management, management reporting, and sharing of data by multiple developers (Amey, 1979). The Ada programming support environment or APSE was suggested as a core set of programming tools consisting of editors, compilers, debuggers, linkers, command languages, and configuration management utilities (Wegner, 1980). Computer aided software engineering was created to automate the tasks of documenting customer requirements, creating software architectures and designs, maintaining requirements traceability, and configuration management (Day, 1983). Integrated computer aided software engineering or I-CASE tools emerged, merging analysis and code generation tools (Banker & Kauffman, 1991).

Software quality assurance. The modern day tenets of software quality assurance began to assume their current form in the late midrange era. Software quality assurance is defined as a "planned and systematic pattern of all actions necessary to provide adequate confidence that the software conforms to established technical requirements" (Abdel-Hamid, 1988). Software quality assurance was created to establish "adherence to coding standards and conventions, compliance with documentation requirements and standards, and successful completion of activities" (Fujii, 1978). Software verification and validation was created to determine the adequacy of software requirements, software designs, software source code, and regression testing during software maintenance (Adrion, Branstad, & Cherniavsky, 1982). Defect prevention was a structured process of determining the root causes of software defects and then institutionalizing measures to prevent their recurrence (Jones, 1985). Quality management systems consisted of a set of organizational policies and procedures to ensure software satisfied its requirements (Rigby, Stoddart, & Norris, 1990).

Software processes. Software processes were formed in the microcomputer era, though they were rooted in the traditions of software engineering, structured methods, software life cycles, and software environments dating back to the late mainframe and early midrange eras. A software process is the "collection of related activities seen as a coherent process subject to reasoning involved in the production of a software system" (Notkin, 1989). The maturity grid (e.g., uncertainty, awakening, enlightenment, wisdom, and certainty), though not for software, inspired the software process modeling movement of the microcomputer era (Crosby, 1979). IBM then created its own process grid (e.g., traditional, awareness, knowledge, skill and wisdom, and integrated management system) for conducting site studies of computer programming laboratories (Radice, Harding, Munnis, & Phillips, 1985). The process maturity framework was

directly adapted from IBM's process grid (Humphrey, 1987), which was finally turned into the capability maturity model for U.S. military use (Weber, Paulk, Wise, & Withey, 1991).

Rapid development. Rapid development was formalized in the microcomputer era though its tenets can be traced back to early notions of structured methods and prototyping. Rapid development is defined as a "methodology and class of tools for speedy object development, graphical user interfaces, and reusable code for client-server applications" (Agarwal, Prasad, Tanniru, & Lynch, 2000). Rapid prototyping was defined as a process of quickly creating an informal model of a software system, soliciting user feedback, and then evolving the model until it satisfied the complete set of customer requirements (Naumann & Jenkins, 1982). Joint application development was a process of having professional software developers assist end users with the development of their applications by evaluating their prototypes (Alavi, 1985). Joint application design, on the other hand, was a structured meeting between end users and software developers with the objective of developing software designs that satisfied their needs (Guide International, Inc., 1986). Rapid systems development was an extension of the joint application design method, which advocated specific technological solutions such as relational databases and the completion of the software system, not just its design (Gane, 1987). In a close adaptation, rapid application development recommended iterative rapid system development cycles in 60 to 120 day intervals (Martin, 1991).

Software reuse. Software reuse assumed its current form in the early microcomputer era, though its earliest tenets can clearly be seen in literature throughout the 1950s, 1960s, and 1970s. Software reuse is the "use of existing software or software knowledge to construct new software" (Frakes & Kang, 2005). Reusable software became synonymous with the Ada programming language in the 1980s, though it was prophesied as a major strategy in 1968 and

was a central management facet of Japanese software factories in the 1970s (Neighbors, 1984). By the close of the microcomputer era, reusable software designs were considered just as important as reusable software source code (Jameson, 1989). The same year, reusability was expanded to include requirements, designs, code, tests, and documents, and dubbed "reusable assets" (Holibaugh, Cohen, Kang, & Peterson, 1989). By the end of the Internet era, catalysis was formed based on composing new applications from existing ones (D'Souza & Wills, 1998).

Software architecture. Software architecture began to assume a strategic role for managing the development of software systems near the end of the microcomputer era. Software architecture is defined as "the structure and organization by which modern system components and subsystems interact to form systems and the properties of systems that can best be designed and analyzed at the system level" (Kruchten, Obbink, & Stafford, 2006). Domain analysis was the discipline of identifying, capturing, and organizing all of the information necessary to create a new software system (Prieto-Diaz, 1987). Domain engineering was a process of managing reusable information about specific types of software systems, gathering architectural data, and gathering data about the computer programs themselves (Arango, 1988). Software architecture was a discipline of creating flexible software designs that were adaptable to multiple computer systems in order to respond to the rapidly changing military threats (Horowitz, 1991). Software product lines soon emerged with an emphasis on evolving software architectures to reduce costs and risks associated with design changes (Wegner, Scherlis, Purtilo, Luckham, & Johnson, 1992), along with software product families (Northrop, 2002).

Agile methods. Agile methods gained prominence in the late Internet and early personalized eras to accommodate the uniquely flexible nature of Internet technologies. Agile methods are an approach for managing the development of software, which is based upon

obtaining early customer feedback on a large number of frequent software releases (Beck, 1999). The dynamic systems development methodology or DSDM has three broad phases that consist of requirements prototypes, design prototypes, and then an implementation or production phase (Millington & Stapleton, 1995). Scrum is a light weight software development process consisting of implementing a small number of customer requirements in two to four week sprint cycles (Schwaber, 1995). Extreme programming or XP consists of collecting informal requirements from on-site customers, organizing teams of pair programmers, developing simple designs, conducting rigorous unit testing, and delivering small and simple software packages in short two-week intervals (Anderson et al., 1998). Open source software development involves freely sharing, peer reviewing, and rapidly evolving software source code for the purpose of increasing its quality and reliability (O'Reilly, 1999). Crystal methods involve frequent delivery; reflective improvement; close communication; personal safety; focus; easy access to expert users; and a technical environment with automated testing, configuration management, and frequent integration (Cockburn, 2002a). Feature driven development involves developing an overall model, building a features list, planning by feature, designing by feature, and building by feature (Palmer & Felsing, 2002). The rational unified process involves a project management, business modeling, requirements, analysis and design, implementation, test, configuration management, environment, and deployment workflow (Kruchten, 2000). Adaptive software development involves product initiation, adaptive cycle planning, concurrent feature development, quality review, and final quality assurance and release (Highsmith, 2000). Lean development involves eliminating waste, amplifying learning, deciding as late as possible, delivering as fast as possible, empowering the team, building integrity in, and seeing the whole (Poppendieck & Poppendieck, 2003).

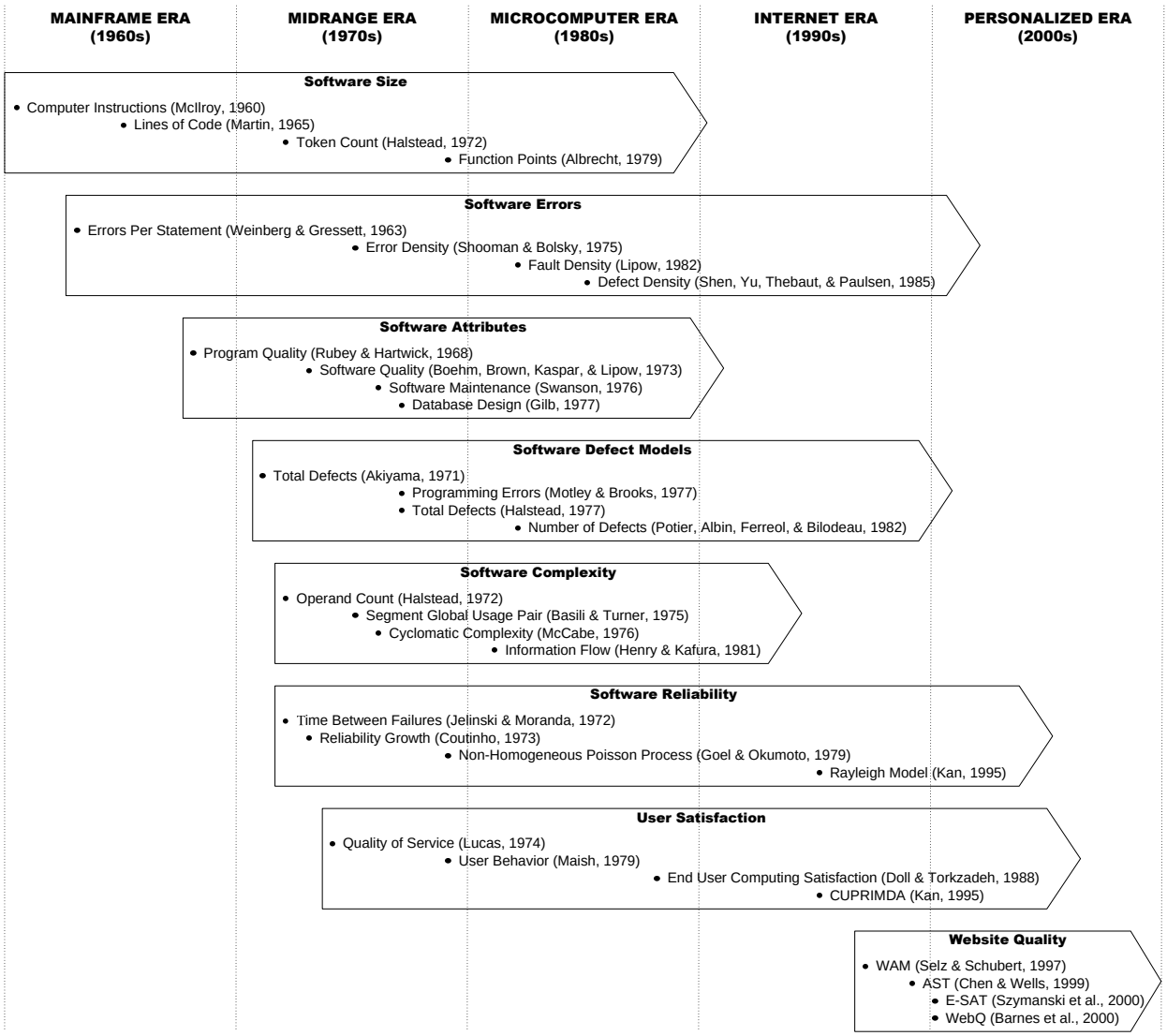

Figure 4. Timeline and history of software quality measures.

Software size. One of the earliest known measures used to describe computer programs was software size (McIlroy, 1960). Software size is a measure of the volume, length, quantity, amount, and overall magnitude of a computer program (Conte, Dunsmore, & Shen, 1986). In the mid-1960s, lines of code or LOC was one of the first known measures of software size, which referred to the number of computer instructions or source statements comprising a computer program and was usually expressed as thousands of lines of code (Martin, 1965). Almost a

decade later in the mid to late 1970s, more sophisticated measures of software size emerged such as token count, volume, function count, and function points (Conte, Dunsmore, & Shen, 1986). Recognizing that individual lines of code had variable lengths, token count was created to distinguish between unequally sized lines of code, which was technically defined as "basic syntactic units distinguishable by a compiler" (Halstead, 1977). In yet another attempt to accurately gauge the size of an individual line of code, volume was created to measure the actual size of a line of code in bits, otherwise known as binary zeros and ones (Halstead, 1977). Shortly thereafter, function count was created to measure software size in terms of the number of modules or subroutines (Basili & Reiter, 1979). Function points was another major measure of software size, which was based on estimating the number of inputs, outputs, master files, inquiries, and interfaces (Albrecht, 1979). Though software size is not a measure of software quality itself, it formed the basis of many measures or ratios of software quality right through the modern era (e.g., number of defects, faults, or failures per line of code or function point). Furthermore, some treatises on software metrics consider software size one of the most basic measures of software complexity (Kan, 1995). Thus, a history of software quality measurement may not be complete without introducing an elementary discussion of software size.

Software errors. One of the earliest approaches for measuring software quality was the practice of counting errors, which dates back to the 1950s when digital computers emerged. Software errors are human actions resulting in defects, defects sometimes manifest themselves as faults, and faults lead to failures, which are often referred to as software crashes (Kan, 1995). The concept of "errors per statement" first appeared in the early 1960s (Weinberg & Gressett, 1963) and studies of "error proneness" intensified by the close of the decade (Youngs, 1970). The term error density was coined in the mid-1970s, which referred to the simple ratio of errors

to software size (Shooman & Bolsky, 1975). Fault density was also a measure of software quality, which referred to the ratio of anomaly causing faults to software size (Lipow, 1982). The term defect density subsumed the measure of error and fault density in the mid-1980s, which referred to the ratio of software errors to software size (Shen, Yu, Thebaut, & Paulsen, 1985). Many unique types of errors were counted, such as number of requirement, design, coding, testing, and maintenance errors, along with number of changes and number of changed lines of code (Conte, Dunsmore, & Shen, 1986). Even the term problem density emerged in the early 1990s, which referred to the number of problems encountered by customers to measure and track software quality (Kan, 1995). The practice of counting errors, defects, faults, and failures as a means of measuring software quality enjoyed widespread popularity for more than five decades.

Software attributes. Another of the earliest approaches for measuring software quality was the practice of quantifying and assessing attributes or characteristics of computer programs. Software attributes are an "inherent, possibly accidental trait, quality, or property" such as functionality, performance, or usability (Institute of Electrical and Electronics Engineers, 1990). Logicon designed a model to measure software attributes such as correctness, logicality, non-interference, optimizability, intelligibility, modifiability and usability (Rubey & Hartwick, 1968). Next, TRW identified software attributes such as portability, reliability, efficiency, modifiability, testability, human engineering, and understandability (Boehm, Brown, Kaspar, & Lipow, 1973). These early works led to numerous specialized spin-offs such as a framework for measuring the attributes of software maintenance (Swanson, 1976) and even a database's design (Gilb, 1977). Spin-offs continued emerging with an increasing focus on operationalizing these attributes with real software metrics (Cavano & McCall, 1978; Dzida, Herda, & Itzfeldt, 1978; Gaffney, 1981). By the mid-1980s, this practice reached Japan (Sunazuka, Azuma, & Yamagishi, 1985) and a

comprehensive framework emerged replete with a detailed software measures (Arthur, 1985). Use of software attributes to measure software quality was exemplified by the functionality, usability, reliability, performance, and supportability or FURPS model (Grady & Caswell, 1987). Software attributes enjoyed widespread use among practitioners throughout the 1970s and 1980s because of their simplicity, though scientists passed them by in favor of statistical models.

Static defect models. One of the earliest approaches for predicting software quality was the use of statistical models referred to as static reliability or static software defect models. "A static model uses other attributes of the project or program modules to estimate the number of defects in software" (Kan, 1995), ignoring "rate of change" (Conte, Dunsmore, & Shen, 1986). One of the earliest software defect models predicted the number of defects in a computer program as a function of size, decision count, or number of subroutine calls (Akiyama, 1971). Multi-linear models were created with up to 10 inputs for the various types of statements found in software code such as comments, data, and executable instructions (Motley & Brooks, 1977). The theory of software science was extended to include defect models by using volume as an input, which itself was a function of program language and statement length (Halstead, 1977). Research into software defect models continued with more extensions based on software science, cyclomatic complexity, path, and reachability metrics (Potier, Albin, Ferreol, & Bilodeau, 1982). More defect models were created by mixing defects, problems, and software science measures such as vocabulary, length, volume, difficulty, and effort (Shen, Yu, Thebaut, & Paulsen, 1985). Later, IBM developed models for predicting problems, fielded defects, arrival rate of problems, and backlog projection, which were used to design midrange operating systems (Kan, 1995). Static linear or multi-linear statistical models to predict defects continue to be useful tools well into modern times, though older dynamic statistical reliability models are overtaking them.

Software complexity. Appearing in the early 1970s, the study of software complexity became one of the most common approaches for measuring the quality of computer programs. Software complexity is defined as "looking into the internal dynamics of the design and code of software from the program or module level to provide clues about program quality" (Kan, 1995). Software complexity sprang from fervor among research scientists eager to transform computer programming from an art into a mathematically based engineering discipline (Halstead, 1972). Many technological breakthroughs in the two decades prior to the mid-1970s led to the formation of software complexity measures. These included the advent of digital computers in the 1950s, discovery of high level computer programming languages, and the formation of compiler theory. Furthermore, flowcharting was routinely automated, axiomatic theorems were used for designing new computer languages, and analysis of numerical computer algorithms became commonplace. As a result, three major classes of software complexity metrics arose for measuring the quality of software: (a) data structure, (b) logic structure, and (c) composite metrics (Weismann, 1973). One of the first data structure metrics was the count of operands, which measured the number of variables, constants, and labels in a computer program versus measuring logic (Halstead, 1972). The segment-global-usage-pair metric determined complexity by counting references to global variables, a high number of which was considered bad among coders (Basili & Turner, 1975). Another data structure metric was the span between variables, which measured how many logic structure statements existed between variables where a higher number was poor (Elshoff, 1976). A unique data structure metric for measuring software quality was the number of live variables within a procedure or subroutine as a sign of undue complexity (Dunsmore & Gannon, 1979). One data structure metric surviving to modern times is the information flow, or fan in - fan out metric, which measures the number of modules that exchange data (Henry & Kafura, 1981).

Logic structure metrics were cyclomatic complexity or paths (McCabe, 1976), minimum paths (Schneidewind & Hoffmann, 1979), and gotos or knots (Woodward, Hennell, & Hedley, 1979). Also included were nesting (Dunsmore & Gannon, 1980), reachability (Shooman, 1983), nest depth (Zolnowsky & Simmons, 1981), and decisions (Shen, Yu, Thebaut, & Paulsen, 1985). Composite metrics combined cyclomatic complexity with other attributes of computer programs to achieve an accurate estimate software quality (Myers, 1977; Hansen, 1978; Oviedo, 1980). They also included system complexity (Card & Glass, 1990) and syntactic construct (Lo, 1992). Finally, it is important to note that most complexity metrics are now defunct, though cyclomatic complexity, which arose out of this era, is still used as a measure of software quality today.

Software reliability. Also emerging in the early 1970s, software reliability was created to predict the number of defects or faults in software as a method of measuring software quality. Software reliability is the "probability that the software will execute for a particular period of time without failure, weighted by the cost to the user of each failure encountered" (Myers, 1976). Major types of reliability models include: (a) finite versus infinite failure models (Musa, 1999), (b) static versus dynamic (Conte, Dunsmore, & Shen, 1986), and (c) deterministic versus probabilistic (Pham, 2000). Major types of dynamic reliability models include: life cycle versus reliability growth (Kan, 1995) and failure rate, curve fitting, reliability growth, non-homogeneous Poisson process, and Markov structure (Pham, 2000). One of the first and most basic failure rate models estimated the mean time between failures (Jelinski & Moranda, 1972). A slightly more sophisticated failure rate model was created based on the notion that software became more reliable with each successive code failure repaired (Schick & Wolverton, 1978). The next failure rate model assumed the failure rate was initially constant and then begins to decrease (Moranda, 1979). Multiple failure rate models appeared throughout the 1970s to round

out this family of reliability models (Goel & Okumoto, 1979; Littlewood, 1979; Sukert, 1979). Reliability or "exponential" growth models followed the appearance of failure rate models, which measured the reliability of computer programs during testing as a function of time or the number of tests (Coutinho, 1973; Wall & Ferguson, 1977). Another major family of reliability models is the non-homogeneous Poisson process models, which estimate the mean number of cumulative failures up to a certain point in time (Huang, 1984; Goel & Okumoto, 1979; Musa, Iannino, & Okumoto, 1987; Ohba, 1984; Yamada, Ohba, & Osaki, 1983). Reliability models estimate the number of software failures after development based on failures encountered during testing and operation. Though rarely mentioned, the Rayleigh life cycle reliability model accurately estimates defects inserted and removed throughout the software lifecycle (Kan, 1995). Some researchers believed the use of software reliability models offered the best hope for transforming computer programming from a craft industry into a true engineering discipline.

User satisfaction. User satisfaction gradually became a measure of software quality during the 1950s, 1960s, and 1970s (Thayer, 1958; Hardin, 1960; Kaufman, 1966; Lucas, 1973). User satisfaction is defined as "the sum of one's feelings or attitudes toward a variety of factors affecting that situation," e.g., computer use and adoption by end users (Bailey & Pearson, 1983). Though not the first, one study of user satisfaction analyzed attitudes toward quality of service, management support, user participation, communication, and computer potential (Lucas, 1974). A more complex study of user satisfaction looked at feelings about staff, management support, preparation for its use, access to system, usefulness, ease-of-use, and flexibility (Maish, 1979). Most studies up until 1980 focused on the end user's satisfaction toward software developers; but one study squarely focused on the end user's satisfaction with the software itself (Lyons, 1980). One of the first studies to address a variety of software attributes such as software accuracy,

timeliness, precision, reliability, currency, and flexibility appeared (Pearson & Bailey, 1980). Studies throughout the 1980s addressed user satisfaction with both designers and software (Walsh, 1982; Bailey & Pearson, 1983; Ives, Olson, & Baroudi, 1983; Joshi, Perkins, & Bostrom, 1986; Baroudi & Orlikowski, 1988). The late 1980s marked a turning point with studies focusing entirely on user satisfaction with the software itself and attributes such as content, accuracy, format, ease-of-use, and timeliness of the software (Doll & Torkzadeh, 1988). A study of user satisfaction at IBM was based on reliability, capability, usability, installability, maintainability, performance, and documentation factors (Kekre, Krishnan, & Srinivasan, 1995). Throughout the 1990s, IBM used a family of user satisfaction models called UPRIMD, UPRIMDA, CUPRIMDA, and CUPRIMDSO, which referred variously to factors of capability, usability, performance, reliability, installability, maintainability, documentation, availability, service, and overall satisfaction (Kan, 1995). User satisfaction, now commonly referred to as customer satisfaction, is no doubt related to earlier measures of software attributes, usability or user friendliness of software, and more recently website quality.

Website quality. Appearing in the late 1990s, following the user satisfaction movement, models of website quality appeared as important measures of software quality (Lindroos, 1997). One of the first models of website quality identified background, image size, sound file display, and celebrity endorsement as important factors of software quality (Dreze & Zufryden, 1997). The web assessment method or WAM quickly followed with quality factors of external bundling, generic services, customer specific services, and emotional experience (Selz & Schubert, 1997). In what promised to be the most prominent web quality model, attitude toward the site or AST had quality factors of entertainment, informativeness, and organization (Chen & Wells, 1999). The next major model was the e-satisfaction model with its five factors of convenience, product

offerings, product information, website design, and financial security (Szymanski & Hise, 2000). The website quality model or WebQual for business school portals was based on factors of ease-of-use, experience, information, and communication and integration (Barnes & Vidgen, 2000). An adaptation of the service quality or ServQual model, WebQual 2.0 measured quality factors such as tangibles, reliability, responsiveness, assurance, and empathy (Barnes & Vidgen, 2001). The electronic commerce user consumer satisfaction index or ECUSI consisted of 10 factors such as product information, consumer service, purchase result and delivery, site design, purchasing process, product merchandising, delivery time and charge, payment methods, ease-of-use, and additional information services (Cho & Park, 2001). Based on nine factors, the website quality or SiteQual model consisted of aesthetic design, competitive value, ease-of-use, clarity of ordering, corporate and brand equity, security, processing speed, product uniqueness, and product quality assurance (Yoo & Donthu, 2001). In what promised to be exclusively for websites, the Internet retail service quality or IRSQ model was based on nine factors of performance, access, security, sensation, information, satisfaction, word of mouth, likelihood of future purchases, and likelihood of complaining (Janda, Trocchia, & Gwinner, 2002). In a rather complex approach, the expectation-disconfirmation effects on web-customer satisfaction or EDEWS model consists of three broad factors (e.g., information quality, system quality, and web satisfaction) and nine subfactors (McKinney, Yoon, & Zahedi, 2002). In one of the smallest and most reliable website quality models to-date, the electronic commerce retail quality or EtailQ model, consists of only four major factors (e.g., fulfillment and reliability, website design, privacy and security, and customer service) and only 14 instrument items (Wolfinbarger & Gilly, 2003). Based on techniques for measuring software quality dating back to the late 1960s, more data has been collected and validated using models of website quality than any other measure.

History of Agile Methods

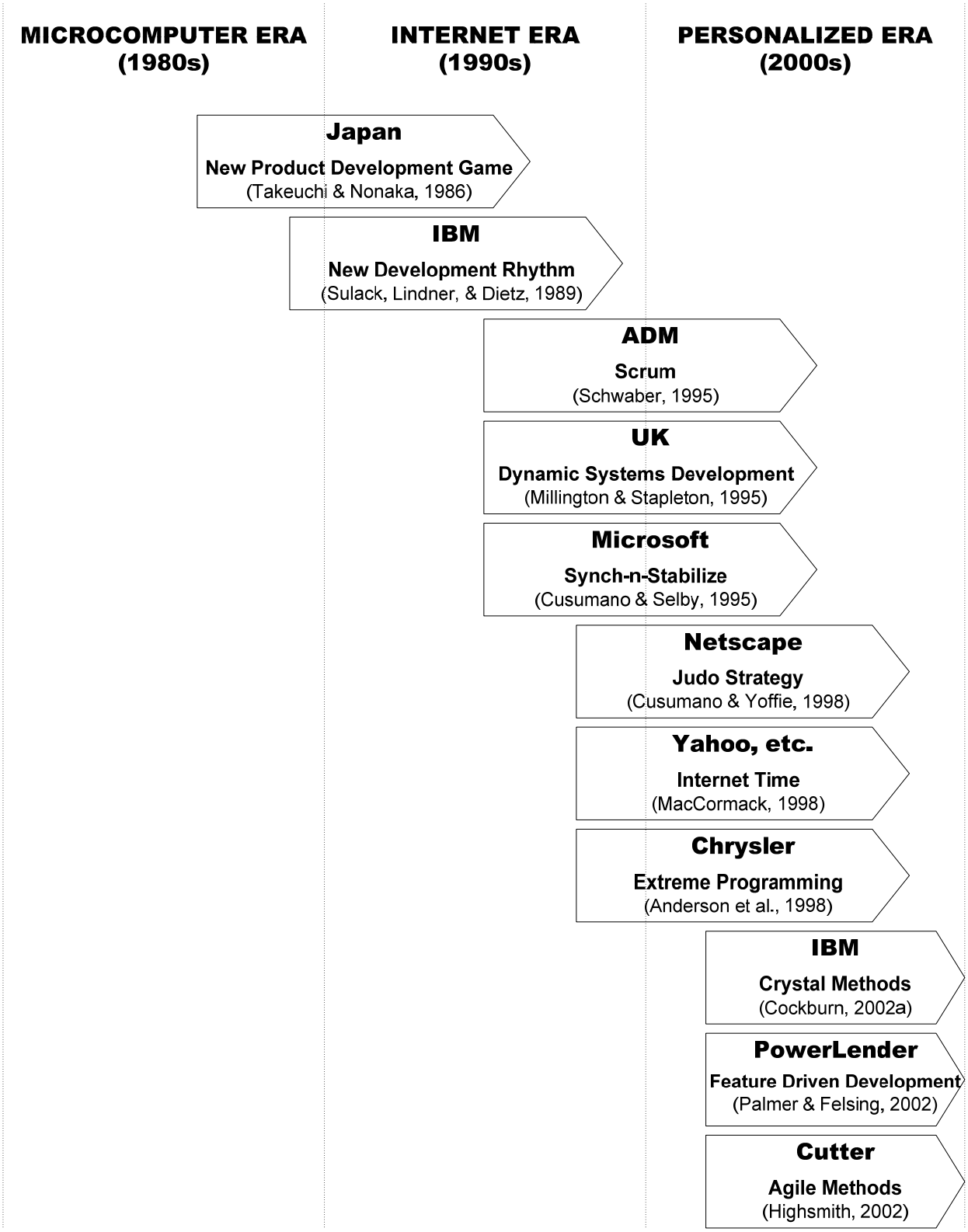

Figure 5. Timeline and history of agile methods.

New product development game. Early in 1986, two management scholars from the School of International Corporate Strategy at Hitotsubashi University in Tokyo, Japan, published a management approach called the "new product development game" in the Harvard Business Review (Takeuchi & Nonaka, 1986). In their article, they argued that Japanese "companies are increasingly realizing that the old sequential approach to developing new products simply will not get the job done." They cited the sport of Rugby as the inspiration for the principles of their new product development game—In particular, Rugby's special play called the Scrum, when the players interlock themselves together as a tightly bound group to gain possession of the ball. The new product development game consisted of six major factors: (a) built-in instability, (b) self organizing project teams, (c) overlapping development phases, (d) multi-learning, (e) subtle control, and (f) organizational transfer of learning. They went on to demonstrate how four Japanese firms, e.g., Fuji-Xerox, Canon, Honda, and NEC, applied the six factors of the new product development game to develop six major products, which became market successes. The six major factors of the new product development game were not unlike the total quality management and concurrent engineering movements that were popular in the U.S. during that timeframe, and their work inspired the development of agile methods for the next 20 years.

New development rhythm. In 1989, three managers from IBM in Rochester, Minnesota, published an article on how IBM devised a management approach called the "new development rhythm," to bring the AS/400 midrange computer to market in only two years (Sulack, Lindner, & Dietz, 1989). In their article, they stated that "user involvement programs yielded a product offering that met the user requirements with a significantly reduced development cycle." The new development rhythm consisted of six major factors: (a) modularized software designs, (b) software reuse, (c) rigorous software reviews and software

testing, (d) iterative development, (e) overlapped software releases, and (f) early user involvement and feedback. IBM's new development rhythm was a remarkable feat of management science and boasted a long list of accomplishments: (a) time-to-market improvement of 40%, (b) development of seven million lines of operating system software in 26 months, (c) compatibility with 30 billion lines of commercial applications, (d) $14 billion in revenues, and (e) the IBM corporation's first Malcolm Baldrige National Quality Award. While there was nothing innovative about IBM's new development rhythm, it was IBM's audacity to apply these academic textbook approaches to commercial product development that was unique.

Scrum. In 1993, Jeff Sutherland of the Easel Corporation adapted the principles from the "new product development game" (Takeuchi & Nonaka, 1986) to the field of computer programming management, explicitly calling it "scrum" (Schwaber, 1995). In particular, scrum assumes that the "systems development process is an unpredictable and complicated process that can only be roughly described as an overall progression." Furthermore, scrum's creators believed "the stated philosophy that systems development is a well understood approach that can be planned, estimated, and successfully completed has proven incorrect in practice." Therefore, scrum's creators set out to define a process as a "loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems." Today, scrum is composed of three broad phases: (a) pre-sprint planning, (b) sprint, and (c) post-sprint meeting. During the pre-sprint planning phase, computer programmers gather to prioritize customer needs. During the sprint phase, computer programmers pretty much do whatever it takes to complete a working version of software that meets a small set of high priority customer needs. Finally, during the post-sprint meeting, computer programmers demonstrate working software to their customers, adjust their priorities, and repeat the cycle.

Dynamic systems development method. In 1993, 16 academic and industry organizations in the United Kingdom banded together to create a management approach for commercial software called the "dynamic systems development method" or simply DSDM (Millington & Stapleton, 1995). Their goal was to "develop and continuously evolve a public domain method for rapid application development" in an era dominated by proprietary methods. Initially, DSDM emphasized three success factors: (a) "the end user community must have a committed senior staff that allows developers easy access to end users," (b) "the development team must be stable and have well established skills," and (c) "the application area must be commercial with flexible initial requirements and a clearly defined user group." These success factors would later be expanded to include functionality versus quality, product versus process, rigorous configuration management, a focus on business objectives, rigorous software testing, risk management, and flexible software requirements. DSDM consists of five major stages: (a) feasibility study, (b) business study, (c) functional model iteration, (d) design and build iteration, and (e) implementation. The goal of DSDM is to explore customer requirements by building at least two full scale prototypes before the final system is implemented.

Synch-n-stabilize. In 1995, two management scholars from MIT's Sloan School of Management published a textbook on how Microsoft managed the development of software for personal computers, dubbed the "sync-n-stabilize" approach (Cusumano & Selby, 1995). Experts on software management approaches for the mainframe market, their two year case study from 1993 to 1995 was more of a grounded theory or emergent research design, which led them to some startling conclusions. At one point in their textbook, they stated that "during this initial research, it became clear why Microsoft was able to remain on top in its industry while most contemporaries from the founding years of the 1970s disappeared." The synch-n-stabilize

approach consisted of six major factors: (a) parallel programming and testing, (b) flexible software requirements, (c) daily operational builds, (d) iterative development, (e) early customer feedback, and (f) use of small programming teams. Microsoft's success was indeed remarkable, and their synch-n-stabilize approach did indeed help them create more than 20 million lines of code for Windows and Office 95, achieve customer satisfaction levels of 95%, and maintain annual profit margins of nearly 36%.

Judo strategy. In 1998, two management scholars from both the Harvard Business School and MIT's Sloan School of Management published a textbook on how Netscape managed the development of software for the Internet, dubbed the "judo strategy" (Cusumano & Yoffie, 1998). Experts on software management approaches for the personal computer market, their one year case study from 1997 to 1998 was once again more of a grounded theory or emergent research design, which prophetically led them to be critical of Netscape's future. Whereas Microsoft's strategic advantage was its immense intellectual capital, Netscape's only advantage seemed to be its first-mover status, which was quickly eroding to Microsoft's market share for browsers at the time their book was published. In fact, the authors criticized Netscape for not having a technical CEO in the fast moving Internet market, which was a very unconventional view among management scholars. Some of the more notable factors characteristic of Netscape's judo strategy included: (a) design products with modularized architectures; (b) use parallel development; (c) rapidly adapt to changing market priorities; (d) apply as much rigorous testing as possible; and (e) use beta testing and open source strategies to solicit early market feedback on features, capabilities, quality, and architecture.

Internet time. In 1998, a management scholar from the Harvard Business School conducted a study on how U.S. firms manage the development of websites, referring to his

approach as “Internet time” (MacCormack, 1998). His study states that “constructs that support a more flexible development process are associated with better performing projects.” Basically, what he did was survey 29 software projects from 15 Internet firms such as Microsoft, Netscape, Yahoo, Intuit, and Altavista. He set out to test the theory that website quality was associated with three major factors: (a) greater investments in architectural design, (b) early market feedback, and (c) greater amounts of generational experience. Harvard Business School scholars believed firms must spend a significant amount of resources creating flexible software designs, they must incorporate customer feedback on working “beta” versions of software into evolving software designs, and higher website quality will be associated with more experience among computer programmers. After determining the extent to which the 29 website software projects complied with these “Internet time” factors through a process of interviews and surveys, he then assembled a panel of 14 industry experts to objectively evaluate the associated website quality. Statistical analysis supported two of the hypotheses, e.g., greater architectural resources and early market feedback were associated with higher website quality, but not the third, e.g., greater experience among computer programmers is associated with higher website quality. This was one of the first studies to offer evidence in support of agile methods.

Extreme programming. In 1998, 20 software managers working for the Chrysler Corporation published an article on how they devised a management approached called “extreme programming” or XP to turn around a failing software project that would provide payroll services for 86,000 Chrysler employees (Anderson et al., 1998). In their article, they stated that “extreme programming rests on the values of simplicity, communication, testing, and aggressiveness.” They also stated that the “project had been declared a failure and all code thrown away, but using the extreme programming methodology, Chrysler started over from

scratch and delivered a very successful result." Extreme programming consists of 13 factors: (a) planning game, (b) small releases, (c) metaphor, (d) simple design, (e) tests, (f) refactoring, (g) pair programming, (h) continuous integration, (i) collective ownership, (j) onsite customer, (k) 40 hour workweek, (l) open workspace, and (m) just rules. What these 20 software managers did was start over, get an informal statement of customer needs, gradually evolve a simple system design using iterative development, apply rigorous testing, use small teams of programmers, and get early customer feedback on their evolving design. In the end, Chrysler was able to deploy an operational payroll system serving more than 86,000 employees.

Crystal methods. In 1991, a software manager with IBM was asked to create an approach for managing the development of object oriented systems called "crystal methods" (Cockburn, 2002a). Crystal methods were piloted on a "$15 million firm, fixed-price project consisting of 45 people." Crystal methods are a "family of methods with a common genetic code, one that emphasizes frequent delivery, close communication, and reflective improvement." Crystal methods are a family of 16 unique approaches for project teams ranging from one to 1,000 people and project criticality ranging from loss of comfort to loss of life. The seven properties of crystal methods are: (a) frequent delivery; (b) reflective improvement; (c) close communication; (d) personal safety; (e) focus; (f) easy access to expert users; and (g) a technical environment with automated testing, configuration management, and frequent integration. The five strategies of crystal methods are: (a) exploratory 360, (b) early victory, (c) walking skeleton, (d) incremental re-architecture, and (e) information radiators. The nine techniques of crystal methods are: (a) methodology shaping, (b) reflection workshop, (c) blitz planning, (d) Delphi estimation, (e) daily stand-ups, (f) agile interaction design, (g) process miniature, (h) side-by-side programming, and (i) burn charts. The eight roles of crystal methods are: (a) sponsor, (b) team

member, (c) coordinator, (d) business expert, (e) lead designer, (f) designer-programmer, (g) tester, and (h) writer. The work products include a mission statement, team structure and conventions, reflection workshop results, project map, release plan, project status, risk list, iteration plan and status, viewing schedule, actor-goal list, use cases and requirements file, user role model, architecture description, screen drafts, common domain model, design sketches and notes, source code, migration code, tests, packaged system, bug reports, and user help text.

Feature driven development. In 1997, three software managers and five software developers created a software development approach called "feature driven development" to help save a failed project for an international bank in Singapore (Palmer & Felsing, 2002). In their textbook, they stated that "the bank had already made one attempt at the project and failed, and the project had inherited a skeptical user community, wary upper management, and a demoralized development team." Furthermore, they stated that "the project was very ambitious, with a highly complex problem domain spanning three lines of business, from front office automation to backend legacy system integration." In order to address this highly complex problem domain that had already experienced severe setbacks, they created an agile and adaptive software development process that is "highly iterative, emphasizes quality at each step, delivers frequent tangible working results, provides accurate and meaningful progress, and is liked by clients, managers, and developers." Feature driven development consists of five overall phases or processes: (a) develop an overall model, (b) build a features list, (c) plan by feature, (d) design by feature, and (e) build by feature. Feature driven development also consists of other best practices in software management and development such as domain object modeling, developing by feature, individual class ownership, feature teams, inspections, regular builds, configuration management, and reporting and visibility of results.

Table 1. Summary of Practices and Processes of Agile Methods

	FDD	Extreme Programming	DSDM	Scrum
Practices	•Domain object modeling •Developing by feature •Class (code) ownership •Feature teams •Inspections •Regular build schedule •Configuration management •Reporting/visibility of results	•Planning game •Small releases •Metaphor •Simple design •Tests •Refactoring •Pair programming •Continuous integration •Collective ownership •On-site customer •40-hour weeks •Open workspace •Just rules	•Active user involvement •Empowered teams •Frequent delivery •Fitness (simplicity) •Iterations and increments •Reversible changes •Baselined requirements •Integrated testing •Stakeholder collaboration	•Product backlog •Burndown chart •Sprint backlog •Iterations and increments •Self managed teams •Daily scrums
Processes	**Develop an Overall Model** Form the Modeling Team Conduct a Domain Walkthrough Study Documents Develop Small Group Models Develop a Team Model Refine the Overall Object Model Write Model Notes Internal and External Assessment **Build a Features List** Form the Features List Team Build the Features List Internal and External Assessment **Plan by Feature** Form the Planning Team Determine Development Sequence Assign Features to Chief Coders Assign Classes to Developers Self Assessment **Iteration (1)** Design by Feature Form a Feature Team Conduct Domain Walkthrough Study Referenced Documents Develop Sequence Diagrams Refine the Object Model Write Class/Method Prologue Design Inspection Build by Feature Implement Classes/Methods Conduct Code Inspection Unit Test Promote to the Build **Iteration (2)** Design by Feature Form a Feature Team Conduct Domain Walkthrough Study Referenced Documents Develop Sequence Diagrams Refine the Object Model Write Class/Method Prologue Design Inspection Build by Feature Implement Classes/Methods Conduct Code Inspection Unit Test Promote to the Build **Iteration (n)** Design by Feature Form a Feature Team Conduct Domain Walkthrough Study Referenced Documents Develop Sequence Diagrams Refine the Object Model Write Class/Method Prologue Design Inspection Build by Feature Implement Classes/Methods Conduct Code Inspection Unit Test Promote to the Build	**User Stories** Requirements Acceptance Tests **Architectural Spike** System Metaphor **Release (1)** Release Planning Release Plan Iteration (1) Iteration Planning Iteration Plan Daily Standup Collective Code Ownership Create Unit Tests Unit Tests Pair Programming Move People Around Refactor Mercilessly Continuous Integration Acceptance Testing Iteration (2) Iteration Planning Iteration Plan Daily Standup Collective Code Ownership Create Unit Tests Unit Tests Pair Programming Move People Around Refactor Mercilessly Continuous Integration Acceptance Testing Iteration (n) Iteration Planning Iteration Plan Daily Standup Collective Code Ownership Create Unit Tests Unit Tests Pair Programming Move People Around Refactor Mercilessly Continuous Integration Acceptance Testing **Release (2)** Iteration (1) Iteration (2) Iteration (n) **Release (n)** Iteration (1) Iteration (2) Iteration (n)	**Feasibility Study** Feasibility Report Feasibility Prototype (optional) Outline Plan Risk Log **Business Study** Business Area Definition Prioritized Requirements List Development Plan System Architecture Definition Updated Risk Log **Functional Model Iteration** Functional Model Functional Prototype (1) Functional Prototype Functional Prototype Records Functional Prototype (2) Functional Prototype Functional Prototype Records Functional Prototype (n) Functional Prototype Functional Prototype Records Non-functional Requirements List Functional Model Review Records Implementation Plan Timebox Plans Updated Risk Log **Design and Build Iteration** Timebox Plans Design Prototype (1) Design Prototype Design Prototype Records Design Prototype (2) Design Prototype Design Prototype Records Design Prototype (n) Design Prototype Design Prototype Records Tested System Test Records **Implementation** User Documentation Trained User Population Delivered System Increment Review Document	**Iteration (1)** Sprint Planning Meeting Product Backlog Sprint Backlog Sprint Daily Scrum Shippable Code Sprint Review Meeting Shippable Code Sprint Retrospective Meeting **Iteration (2)** Sprint Planning Meeting Product Backlog Sprint Backlog Sprint Daily Scrum Shippable Code Sprint Review Meeting Shippable Code Sprint Retrospective Meeting **Iteration (n)** Sprint Planning Meeting Product Backlog Sprint Backlog Sprint Daily Scrum Shippable Code Sprint Review Meeting Shippable Code Sprint Retrospective Meeting

History of Studies on Agile Methods

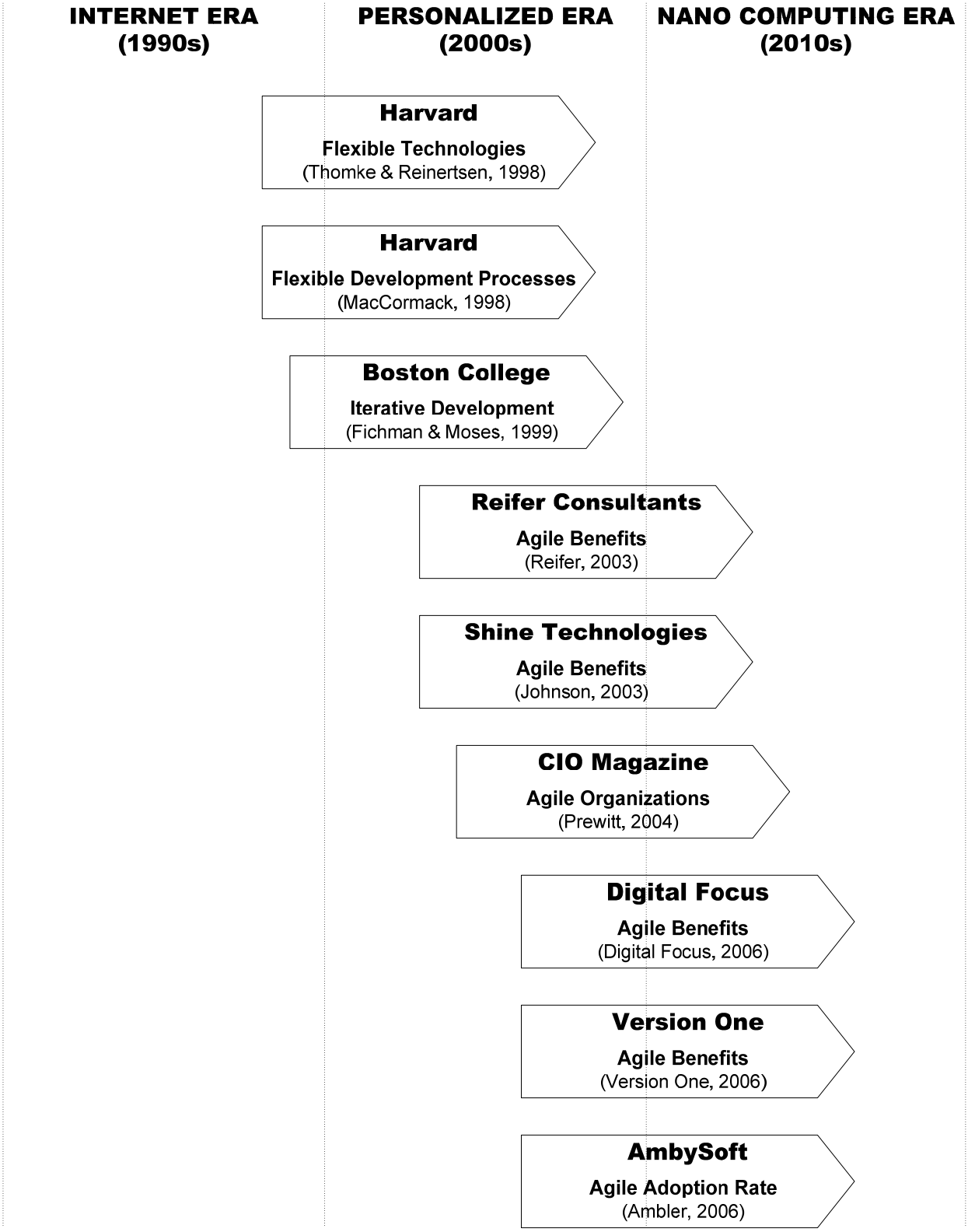

Figure 6. Timeline and history of studies on agile methods.

Harvard business school. In 1998, two management scholars from the Harvard Business School conducted a survey of 391 respondents to test the effects of flexible versus inflexible product technologies (Thomke & Reinertsen, 1998). What they found was that projects using inflexible product technologies required over two times as much engineering effort as flexible product technologies (e.g., 17.94 vs. 8.15 months).

Harvard business school. In 1998, a management scholar from the Harvard Business School conducted a survey of 29 projects from 15 U.S. Internet firms to test the effects of flexible software development management approaches on website quality (MacCormack, 1998). What he found was that flexible product architectures and customer feedback on early beta releases were correlated to higher levels of website quality.

Boston college carroll school of management. In 1999, two management scholars from Boston College's Carroll School of Management conducted a case study of 28 software projects to determine the effects of iterative development on project success (Fichman & Moses, 1999). What they found was that software projects that use iterative development deliver working software 38% sooner, complete their projects twice as fast, and satisfy over twice as many software requirements.

Reifer consultants. In 2003, Reifer Consultants conducted a survey of 78 projects from 18 firms to determine the effects of using agile methods to manage the development of software (Reifer, 2003). What they found was that 14% to 25% of respondents experienced productivity gains, 7% to 12% reported cost reductions, and 25% to 80% reported time-to-market improvements.

Shine technologies. In 2003, Shine Technologies conducted an international survey of 131 respondents to determine the effects of using agile methods to manage the development of

software (Johnson, 2003). What they found was that 49% of the respondents experienced cost reductions, 93% of the respondents experienced productivity increases, 88% of the respondents experienced quality increases, and 83% experienced customer satisfaction improvements.

CIO magazine. In 2004, CIO Magazine conducted a survey of 100 information technology executives with an average annual budget of $270 million to determine the effects of agile management on organizational effectiveness (Prewitt, 2004). What they found was that 28% of respondents had been using agile management methods since 2001, 85% of the respondents were undergoing enterprise wide agile management initiatives, 43% of the respondents were using agile management to improve organizational growth and market share, and 85% said agile management was a core part of their organizational strategy.

Digital focus. In 2006, Digital Focus conducted a survey of 136 respondents to determine the effects of using agile methods to manage the development of software (Digital Focus, 2006). What they found was that 27% of the respondents were adopting agile methods for a project, 23% of the respondents were adopting agile methods company wide, 51% of the respondents wanted to use agile methods to speed up the development process, 51% of the respondents said they lacked the skills necessary to implement agile methods at the project level, 62% of the respondents said they lacked the skills necessary to implement agile methods at the organization level, and 60% planned on teaching themselves how to use agile methods.

Version one. In 2006, Version One conducted an international survey of 722 respondents to determine the effects of using agile methods to manage the development of software (Version One, 2006). What they found was that 86% of the respondents reported time-to-market improvements, 87% of the respondents reported productivity improvements, 86% of the respondents reported quality improvements, 63% of the respondents reported cost reductions,

92% of the respondents reported the ability to manage changing priorities, 74% of the respondents reported improved morale, 72% of the respondents reported risk reductions, 66% of the respondents reported satisfaction of business goals, and 40% were using the scrum method.

Ambysoft. In 2006, Ambysoft conducted an international survey of 4,232 respondents to determine the effects of using agile methods to manage the development of software (Ambler, 2006). What they found was that 41% of organizations were using agile methods; 65% used more than one type of agile method; 44% reported improvements in productivity, quality, and cost reductions; and 38% reported improvements in customer satisfaction.

Table 2. Summary of Recent Studies and Surveys of Agile Methods

Year	Source	Findings	Responses
1998	Harvard (Thomke et al., 1998)	50% reduction in engineering effort 55% improvement in time to market 925% improvement in number of changes allowed	391
1998	Harvard (MacCormack, 1998)	48% productivity increase over traditional methods 38% higher quality associated with more design effort 50% higher quality associated with iterative development	29
1999	Boston College (Fichman et al., 1999)	38% reduction in time to produce working software 50% time to market improvement 50% more capabilities delivered to customers	28
2003	Reifer Consultants (Reifer, 2003)	20% reported productivity gains 10% reported cost reductions 53% reported time-to-market improvements	78
2003	Shine Technologies (Johnson, 2003)	49% experienced cost reductions 93% experienced productivity increases 88% experienced customer satisfaction improvements	131
2004	CIO Magazine (Prewitt, 2004)	28% had been using agile methods since 2001 85% initiated enterprise-wide agile methods initiatives 43% used agile methods to improve growth and marketshare	100
2006	Digital Focus (Digital Focus, 2006)	27% of software projects used agile methods 23% had enterprise-wide agile methods initiatives 51% used agile methods to speed-up development	136
2006	Version One (Version One, 2006)	86% reported time-to-market improvements 87% reported productivity improvements 92% reported ability to dynamically change priorities	722
2006	AmbySoft (Ambler, 2006)	41% of organizations used agile methods 44% reported improved productivity, quality, and costs 38% reported improvements in customer satisfaction levels	4,232

Gaps and Problem Areas in the Literature

The gaps and problem areas in the literature associated with agile methods and website quality are numerous. First, there are few scholarly studies of agile methods. That is, this author has been unable to locate and identify a single scholarly study containing a theoretical conceptual model of agile methods. Furthermore, not a single article in the literature review was based on a systematic qualitative or quantitative study of agile methods. The literature review only mentions textbooks and articles with notional concepts in agile methods. Even the quantitative survey research mentioned in the literature review was of a rudimentary attitudinal nature. Additionally, none of the articles mentioned in the literature review addressed all four of the factors associated with agile methods (e.g., iterative development, customer feedback, well-structured teams, and flexibility). None of them were systematically linked to scholarly models of website quality. The gaps are quite clear, a dearth of holistic scholarship exists on agile methods.

Need for a New Study of Agile Methods

There is a clear need for a new study of agile methods. Once again, we hope to develop one of the first in a long line of systematic scholarly studies of agile methods. Furthermore, we will attempt to link the factors of agile methods to scholarly models of website quality. First and foremost, there is a need for a systematic analysis of scholarly literature associated with the factors of agile methods, which the next section will attempt to do. Then there is a need for a scholarly theoretical model of agile methods, depicting the factors, variables, and hypotheses associated with using agile methods. Additionally, there is a need for an analysis of scholarly literature to identify the factors and variables associated with website quality. Finally, there is a need to identify, survey, select, or develop scholarly measures and instruments for both agile methods and website quality. Together these would constitute a new study of agile methods.

CONCEPTUAL FRAMEWORK

The purpose of this section is to present a framework for examining the links between the factors of agile methods and website quality among electronic commerce firms. This section identifies four major factors of agile methods from an analysis of relevant literature: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. In total, nine major agile methods were analyzed: (a) new development rhythm, (b) scrum, (c) dynamic systems development, (d) synch-n-stabilize, (e) Internet time, (f) judo strategy, (g) extreme programming, (h) feature driven development, and (i) open source software development. The conceptual model in Figure 7 represents the results of the analysis of factors and subfactors of agile methods and their relationships. Furthermore, four hypotheses were formulated linking the factors of agile methods to website quality. The conceptual model is a graphical illustration of the goals, scope, and boundaries of this study.

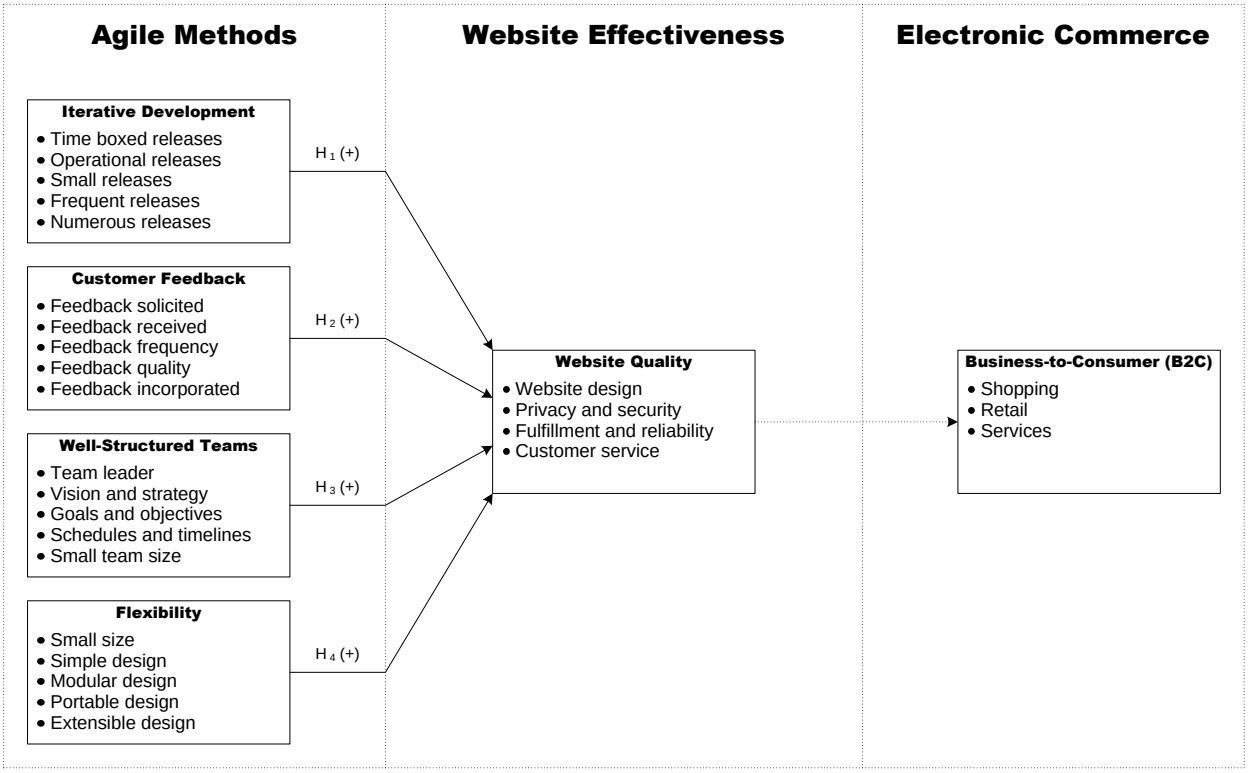

Figure 7. Conceptual model of agile methods, website quality, and e-commerce.

Agile methods. Although the tenets of agile methods have been in place for as long as five decades, modernized agile methods have only been in existence for about four or five years. Subsequently, few scholarly definitions of agile methods have been produced or exist. However, there are numerous informal definitions of agile methods, which will be examined here. “Agility is a comprehensive response to the business challenges of profiting from rapidly changing and continually fragmenting global markets for high quality, high-performance, and customer-configured goods and services” (Goldman, Nagel, & Preiss, 1995). “Agility is the ability to both create and respond to change in order to profit in a turbulent business environment” (Highsmith, 2002). “Agile development methods apply time-boxed iterative and evolutionary development, adaptive planning, evolutionary delivery, and other values and practices to encourage rapid and flexible response to change” (Larman, 2004). “Agility is the ability to deliver customer value while dealing with inherent project unpredictability and dynamism by recognizing and adapting to change” (Augustine, 2005). “Agile software development promotes quick response to changes in requirements as well as extensive and ongoing collaboration between the development team and the customer” (Germain & Robillard, 2005). “Agile methods are characterized by short iterative cycles of development driven by product features, periods of reflection and introspection, collaborative decision making, incorporation of rapid feedback and change, and continuous integration of code changes into the system under development” (Nerur, Mahapatra, & Mangalaraj, 2005). “Agile programming is design for change without refactoring and rebuilding, its objective is to design programs that are receptive to and expect change, and it lets changes be applied in a simple localized way to avoid or substantially reduce major refactoring, retesting, and system builds” (Thomas, 2005). “Agile denotes the quality of being agile, ready for motion, nimble, active, and dexterous in motion, which is what software development

methods are attempting to offer in order to answer the eager business community asking for lighter weight, faster, and nimbler software development processes" (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). "Being agile is a declaration of prioritizing for project maneuverability with respect to shifting requirements, shifting technology, and a shifting understanding of the situation" (Cockburn, 2002b). "Agile methods are human-centric bodies of practices and guidelines for building usable software in unpredictable and highly-volatile environments; they encourage continual realignment of development goals with the needs and expectations of the customer; and they concentrate on significantly improving communications and interactions among team members and with the customer, promoting continuous feedback, focusing on clean code that works, transparency, and merciless testing to achieve higher quality" (Melnik & Maurer, 2005). "Agile processes focus on the early facilitation and fast production of working code, and are based on software development process models that support iterative and incremental development of software" (Turk, France, & Rumpe, 2005). "Agile methods are a better way of developing software, which involves working software over comprehensive documentation, customer collaboration over contract negotiation, individuals and interactions over processes and tools, and responding to change over following a plan" (Agile Manifesto, 2001). These 12 definitions reveal a diversity of interrelated concepts with respect to agile methods. For the purposes of this research framework, we conclude that agile methods are processes for developing new products in which small, but structured teams solicit and incorporate early market feedback into a stream of rapid, frequent, and numerous releases of working software that are intentionally designed to evolve, grow, and change in order to discover and satisfy customer needs and produce high quality software.

Iterative development. Iterative development was invented in the 1970s, sometimes used in the 1980s, gained momentum in the 1990s, and is accepted as a coding principle today. There have been many variations of iterative development, such as incremental, evolutionary, and spiral development, as well as experimentation, learning by doing, and sense and response. The notion of iterative development is that product architecture begins with a simple design and then evolves into a more complex design as data is learned about technology and market needs. There are basically two approaches to design: (a) gather all user needs at once and then design the product all at one time, or (b) evolve designs a little at a time as user needs are discovered. The former is called scope-boxed development and the latter is called time-boxed development, both of which have fixed delivery schedules. There is a subtle but important difference between the two and a few other misconceptions about iterative development. First, traditional scope-boxed development is particularly risky, because if the scope is too large or too complex then one of three situations will arise: (a) the schedule will slip, (b) the budget will overrun, or (c) the contract will enter into default. In time-boxed development, none of these things happen, because the requirements may be relaxed (e.g., reduced), in order to meet the schedule, budget, or contract stipulations. Other common misconceptions about iterative development are that one can develop requirements and designs in increments, and then code the software all at once in the end. A more important caveat is that each increment must be operational software. Even more importantly, there must be many operational increments, not just a few. A major study of new product development from the 1990s lauded that the best product designers developed an operational product in five years, while market laggards developed products in seven or more. The rapid application development movement of the early 1990s yielded operational software in three, six, nine, twelve, eighteen, and even twenty-four month intervals. Mainstream agile

methods yielded their products in two, four, and twelve week increments. Internet time, synch-n-stabilize, and open source software development methods yielded operational increments on a daily basis. A minor flaw in scholarly studies of iterative development was their reliance on only two or three operational increments in total. Many have ignored earlier research indicating that the difference between unsuccessful and successful products was 14 operational increments. Therefore, the following five subfactors of iterative development have been selected from the literature on agile methods: (a) time-boxed releases, (b) operational releases, (c) small releases, (d) frequent releases, and (e) numerous releases.

Customer feedback. The field of customer feedback as it relates to agile methods seems to be bifurcated. Most of the agile methods from the 1990s equate customer feedback to the participation by one's customers in every aspect of project decision making. However, smaller, leaner, and more effective agile methods equate customer feedback to early market feedback on working or operational software. In the latter case, customers are not involved in project decision making, but do give rather extensive evaluations or suggestions for improvement on working code. This is a most interesting finding, because most academic software methods favor the over-involved, micromanaging customer. On the other hand, most industry models of agile methods used by Internet firms prefer the model of early market feedback (i.e., "I will give you a beta release, but my security guard will stop you at the door if you try to see me"). Therefore, the following five subfactors of customer feedback have been selected from the literature on agile methods: (a) feedback solicited, (b) feedback received, (c) feedback frequency, (d) feedback quality, and (e) feedback incorporated.

Well-structured teams. Well-structured teams have historically been regarded as small groups of workers who are responsible for accomplishing their tasks with little or no supervision.

Early studies demonstrated the ability of well-structured teams to satisfy their goals and objectives without the necessity of having task masters who were responsible for efficiency of employees. However, the agile manifesto's creators stated that well-structured teams within agile methods were not to be mistaken with the historical principles of self-managing or self-directed teams. Instead, agile methods give managers the responsibility and authority to set directions, establish boundaries, build multi-tiered structures, and hire programmers to carry out their decisions. These principles are very similar to more recent research on self-organizing, self-managing, and self-directed teams. That is, studies have shown that teams with clear leaders, goals, objectives, plans, tasks, and timelines, no matter how formal or informal, perform better. An analysis of the literature on agile methods from the 1990s shows that successful projects have had varying degrees of formal management structures calling for strategic vision to be mixed together with small teams who were responsible for producing releases of working software as rapidly as possible, even on a daily basis. Therefore, the following five subfactors of well-structured teams have been selected from the literature on agile methods: (a) team leader, (b) vision and strategy, (c) goals and objectives, (d) schedules and timelines, and (e) small team size.

Flexibility. The fourth value found in the agile manifesto is "responding to change." An analysis of the literature on agile methods reveals this value was meant to embody the theory of "adaptable organizations" and represented an entire genre of popular literature equating organizations to living organisms, evolutionary biology, and survival of the fittest. In actuality, the notion of organizations as organisms was conceived in the 1940s, gained notoriety with the rise of systems dynamics in the 1960s, and entered into mainstream consciousness in the 1990s. So, one could legitimately label the fourth value of agile methods "adaptability." However, an analysis of the literature on agile methods reveals that in every case the fourth major factor of

agile methods was not adaptability, but rather the flexibility of software architectures and designs. That is, in order to succeed with iterative development, software architectures, designs, and code had to be proactively structured in such a way as to accommodate continuous change or replacement. Therefore, the last major factor of agile methods has been named "flexibility" and the following five subfactors have been selected from the literature on agile methods: (a) small size, (b) simple design, (c) modular design, (d) portable design, and (e) extensible design.

Website quality. Since the 1960s, increasingly sophisticated views of software quality have emerged: (a) software size, (b) software errors, (c) software attributes, (d) software defect models, (e) software complexity, (f) software reliability, (g) user satisfaction, and (h) website quality. One of the earliest approaches for measuring software quality was the practice of quantifying and assessing attributes or characteristics of computer programs. Early studies attempted to enumerate, qualify, and quantify all of the attributes of software products. One such study identified the following attributes: (a) correctness, (b) efficiency, (c) flexibility, (d) integrity, (e) interoperability, (f) maintainability, (g) portability, (h) reliability, (i) reusability, (j) testability, and (k) usability. During the 1990s, user satisfaction models were used to measure end user attitudes towards software products. One such model measured user attitudes about the following attributes of software quality: (a) capability, (b) usability, (c) performance, (d) reliability, (e) installability, (f) maintainability, (g) documentation, (h) availability, (i) service, and (j) overall satisfaction. Models of user satisfaction were eventually overtaken by models of website quality by the end of the 1990s. Basic website quality is defined as a "customer's judgment about the website's overall excellence or superiority, which is an attitude that comes from a comparison of expectations and perceived performance" (Arambewela & Hall, 2006). Within the context of electronic commerce, website quality refers to "the extent to which a

website facilitates efficient and effective shopping, purchasing, and delivery of products and services" (Gounaris, Dimitriadis, & Stathakopoulos, 2005). Over 45 scholarly models of website quality have appeared in the last 10 years. A small sample of those studies had been tested on over 436,000 data points from 16,000 respondents. What this indicates is that the application and use of scholarly models of website quality is a very-well established discipline. However, many of these models have numerous factors and subfactors, as well as unusually large measurement instruments, which are economically prohibitive to apply. Many of these models have not proven very robust, and exhibit low levels of reliability and validity. Since the purpose of this study is to determine the effects of agile methods on website quality, we propose to use the eTailQ model to measure website quality. The eTailQ model was extensively tested on over one thousand respondents and exhibits high levels of reliability and validity (Wolfinbarger & Gilly, 2003). The eTailQ model of website quality consists of four major subfactors: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service.

Table 3. Major Factors of Website Quality from an Analysis of Major Approaches

Method	Major Factors
WAM	Product/service system, external bundling, generic services, customer-specific services, emotional customer experience
AST	Entertainment, informativeness, organization
E-Satisfaction	Convenience, merchandizing, website design, financial security
WebQual	Tangibles, reliability, responsiveness, assurance, empathy
ECUSI	Product information, consumer service, purchase result and delivery, website design, purchasing process, product sales, delivery time and charge, payment methods, ease-of-use
SiteQual	Ease-of-use, aesthetic design, processing speed, security
IRSQ	Performance, access, security, sensation, information
EDEWS	Expectation, perceived performance, disconfirmation, satisfaction
e-SQ	Customer website requirements, customer website experiences, perceived quality, perceived value, purchase/repurchase
eTailQ	Website design, privacy and security, fulfillment and reliability, customer service

Major Factors of Agile Methods

Since the 1980s, the factors of new product development have been adapted to software methods to produce innovatively new computer software. The new development rhythm emphasized early user involvement, iterative processes, cross-functional teams, and modularity. Scrum emphasized stakeholder feedback, iterative development, self-managed teams, and early architectural design. DSDM emphasized user involvement and stakeholder cooperation, iterative development, empowered teams, and simple flexible designs. Synch-n-stabilize emphasized continuous customer feedback, iterations, small teams, and evolving specifications. Internet time emphasized market feedback, prototypes and beta versions, experienced teams, and architectural design. The judo strategy emphasized market feedback, beta testing, small teams, and cross-platform designs. All of the known agile methods, including XP, FDD, OSS, and the agile manifesto have four major factors in common: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility.

Table 4. Major Factors of Agile Methods from an Analysis of Major Approaches

Method	Major Factors
New development rhythm	Iterations [1], involvement [2], empowered teams [3], modularity [4], synchronization, configuration control, dependency management, performance reviews, metrics, testing, reviews
Scrum	Iterative development [1], stakeholder feedback [2], self managing teams [3], prioritized requirements [3], daily team meetings [3], early architectural design [4]
Dynamic systems development	Iterative development [1], frequent delivery [1], user involvement [2], stakeholder cooperation [2], empowered teams [3], simple flexible designs [4], change control, high-level requirements, tests
Synch-n-stabilize	Iterations [1], daily builds [1], releases [1], customer feedback [2], small teams [3], vision statements [3], prioritized features [3], milestones [3], evolving specifications [4], parallel development
Internet time	Rapid prototyping and early beta releases [1], daily incorporation of rapid market feedback [2], experienced teams [3], large investments in software architecture and design [4]
Judo strategy	Beta testing [1], market feedback [2], small teams [3], cross platform design [4], modular designs [4], reuse [4], flexible priorities [4], evolving features [4], parallel development, testing
Extreme programming	Releases [1], on-site customer [2], pair programming [3], simplicity [4], planning, metaphors, tests, refactoring, continuous integration, collective owners, 40-hours, open workspace, just rules
Feature driven development	Regular builds [1], domain experts [2], feature teams [3], technical architecture [4], object modeling, design by feature, class (code) ownership, inspections, configuration management, reporting
Open source software	Rapid releases [1], increased user involvement [2], prompt feedback [2], international community [3], highly-talented developers [3], evolutionary designs [4], parallel development, peer reviews
Agile manifesto	Working software [1], Customer collaboration [2], individuals and interactions [3], responding to change [4]

[1] *Iterative development* — [2] *Customer feedback* — [3] *Well-structured teams* — [4] *Flexibility*

New development rhythm. In 1988, IBM's computer division in Minnesota created a new product development process called the "silverlake project" (Bauer, Collar, & Tang, 1992). No doubt inspired by the new product development game (Takeuchi & Nonaka, 1986) and Peter Drucker, IBM created the silverlake project to help turnaround a failed project called fort knox. The silverlake project consisted of 10 major factors: (a) visionary leaders, (b) talented people, (c) empowerment, (d) cross-functional teams, (e) market segmentation, (f) market analysis, (g) setting priorities, (h) parallel processes, (i) customer partnerships, and (j) customer satisfaction. Likewise, IBM devised a software method called the new development rhythm, which consisted of 17 major factors: (a) walkthroughs, (b) inspections, (c) iterative development, (d) early user involvement, (e) user feedback, (f) design synchronization, (g) flatter organizations, (h) configuration control, (i) design control, (j) dependency management, (k) quality management, (l) performance reviews, (m) reliability modeling, (n) formal testing, (o) empowerment, (p) cross-functional teams, and (q) modular designs (Sulack, Lindner, & Dietz, 1989). IBM combined the major factors of the silverlake project and new development rhythm to consolidate their midrange computers in only two years, which was half of the normal development time. During this time, IBM reused five million lines of code, designed two million lines of new code, and helped their customers port 30 billion lines of code to the new computer system (Pine, 1989). The new development rhythm was directly linked to high customer satisfaction (Kan, Dull, Amundson, Lindner, & Hedger, 1994) and IBM's stock market performance (Hoisington, 1998). IBM used the major factors of the silverlake project and new development rhythm to generate $14 billion in revenues and also help IBM win its first baldrige award (Tang & Collar, 1992). IBM's adaptation of the new product development game formed a broad framework and pattern, which would be used to describe innovative software processes throughout the 1990s.

Scrum. In 1993, Jeff Sutherland of the Easel Corporation created an extremely simple software method called scrum to complete the design of a software package (Sutherland, 2004). Scrum was based on the six major factors of the new product development game: (a) built-in instability, (b) self organizing project teams, (c) overlapping development phases, (d) multi-learning, (e) subtle control, and (f) organizational learning transfer (Takeuchi & Nonaka, 1986). Scrum was based on six major factors: (a) iterative development, (b) prioritized requirements, (c) early architectural design, (d) daily team meetings, (e) self managing teams, and (f) stakeholder feedback (Schwaber, 2004). Scrum was comprised of five major stages or processes: (a) sprint planning meeting, (b) sprint, (c) daily scrum meetings, (d) sprint review meetings, and (e) sprint retrospective meetings (Schwaber, 2004). During the sprint cycle, teams met on a daily basis to report how much code they had completed, not just work-in-progress, which was considered a sign of weakness (Schwaber, 2004). Scrum was also comprised of four major roles: (a) product owner, (b) team members, (c) scrum master, and (d) stakeholders (Schwaber, 2004). In scrum, the first three roles were comprised of people who were committed to a project's success, while stakeholders, who represent customers and end users were not (Schwaber, 2004). Scrum masters were people trained to teach and administer the scrum method. Customer collaboration and design occurred during sprint planning and sprint review meetings, iterative development was accomplished by sprints, small cross-functional teams came together in daily scrums, and software refactoring occurred during the sprints as well (Schwaber, 2004). Though scrum was based on knowledge creation theory, it seemed to get its strength from the power and synergy of iterative development and self-managed teams rather than early customer feedback (Schwaber, 2004). Today, there are more than 11,000 certified scrum masters and it is growing in popularity due to its overall simplicity and low-market entry costs contrary to traditional methodologies.

Dynamic systems development method. Also in 1993, a British consortium formed to create the dynamic systems development method or DSDM (Millington & Stapleton, 1995). The goal of the DSDM consortium was to create a publicly available software development methodology to counter proprietary rapid application development methods formed in the 1990s. DSDM consisted of nine major factors: (a) user involvement, (b) team empowerment, (c) frequent delivery, (d) fitness for use, (e) iterative development, (f) change control, (g) high-level requirements, (h) thorough testing, and (i) stakeholder cooperation (DSDM Consortium, 1995). DSDM consisted of five major stages: (a) feasibility study, (b) business study, (c) functional model iteration, (d) system design and build iteration, and (e) implementation. DSDM was also comprised of 15 major practices: (a) time-boxing, (b) daily meetings, (c) requirements prioritization, (d) project management, (e) escalation management, (f) project planning, (g) quality management, (h) risk management, (i) estimating, (j) facilitated workshops, (k) modeling, (l) prototyping, (m) testing, (n) configuration management, and (o) tool support environments. DSDM consisted of 12 roles: (a) executive sponsor, (b) visionary, (c) ambassador user, (d) advisor user, (e) project manager, (f) technical coordinator, (g) team leader, (h) developer, (i) tester, (j) facilitator, (k) scribe, (l) and specialist. DSDM consisted of 23 work products: business area definition, delivered system, design prototype, design prototyping review records, development plan, feasibility prototype, feasibility report, functional model, functional model review records, functional prototype, implementation plan, increment review document, non-functional requirements list, outline plan, post-implementation review report, prioritized requirements list, risk log, system architecture definition, tested system, test records, time-box plan, trained user population, and user documentation. DSDM has not enjoyed the widespread use in the market place as other agile methods, but continues to have a noticeable following.

Synch-n-stabilize. Microsoft grew from three employees and $16,000 in revenue in 1975 to almost 18,000 employees and $6 billion in revenue by 1995 (Cusumano & Selby, 1995). A seven-pronged strategy called "Microsoft's secrets" was credited with helping them achieve this phenomenal growth and create seven operating systems and 13 products in this time frame. The seven factors of Microsoft's secrets were: (a) find smart people who know the technology and business; (b) organize small teams of overlapping functional specialists; (c) pioneer and orchestrate evolving mass markets; (d) focus creativity by evolving features and fixing resources; (e) do everything in parallel with frequent synchronizations; (f) improve through continuous self-critiquing, feedback, and sharing; and (g) attack the future. Microsoft's software development method, dubbed "synch-n-stabilize," consisted of seven major factors as well: (a) product development and testing done in parallel, (b) vision statement and evolving specification, (c) features prioritized and built in 3 or 4 milestone subprojects, (d) frequent synchronizations (daily builds) and intermediate stabilizations (milestones), (e) fixed release and ship dates and multiple release cycles, (f) continuous customer feedback in the development process, and (g) product and process design so that large teams work like small teams. The key to synch-n-stabilize was its daily build process, which consisted of 11 major techniques: (a) check out, (b) implement feature, (c) build private release, (d) test private release, (e) synch code changes, (f) merge code changes, (g) build private release, (h) test private release, (i) execute quick test, (j) check in, and (k) generate daily build. Buried deep within Microsoft's process were 14 kinds of software tests: usage, interface, ad hoc, 16-bit application, gorilla, user interface, stress, 32-bit application, verification, applets, independent, bug bash, simulation, automated, and various other types of tests. Nearly 30 years after its inception, Microsoft's corporate culture was still regarded as free wheeling, informal, individualistic, highly-motivated, and innovative (Herbold, 2002).

Internet time. The term “Internet time” emerged in the mid 1990s and referred to the instantaneous speed at which beta versions of web browsers were distributed (Lundquist, 1996). The term Internet time became synonymous with Silicon Valley startups like Netscape, Yahoo, and Alta Vista, because of their daily releases of browsers, email services, and search engines. Even Microsoft opened the floodgates on their synch-n-stabilize process to allow beta versions of their web browsers to be released on a nightly basis allowing them to catch up with Netscape. Because of the success of Internet startups, Internet time became the new product development process of the late 1990s and came under scrutiny by scholars from Ivy League business schools. One such study characterized Internet time in terms of three broad factors: (a) testing technical solutions, (b) sensing the market, and (c) integrating customer needs with technical solutions (Iansiti & MacCormack, 1997). Testing technical solutions referred to building rapid prototypes and beta versions, sensing the market referred to soliciting market feedback, and integrating customer needs referred to acting on market feedback. One of the first empirical studies of Internet time identified four major factors: (a) an early release of the evolving product design to customers, (b) daily incorporation of new software code and rapid feedback on design changes, (c) a team with broad-based experience shipping multiple projects, and (d) major investments in the design of the product architecture (MacCormack, 2001). This study revealed a correlation between: (a) gradual evolution using beta versions and increasing quality, (b) number of beta versions and increasing quality, and (c) shorter market feedback cycles and increasing quality. Later studies showed that Internet time required as much as 25% of the total project effort hours to be dedicated to architectural and design activities (MacCormack, Verganti, & Iansiti, 2001). Further analysis of the data shows that productivity rates of Internet time were an order of magnitude better than historical averages and 50% better than state-of-the-art techniques.

Judo strategy. Netscape grew from two employees and $2 million in revenues in 1994 to more than 1,500 employees and $440 million in revenues by 1998 (Buckley, Tse, Rijken, & Eijgenhuijsen, 2002). Netscape's operating philosophy for competing with Microsoft was dubbed the "Judo Strategy" by management scholars from MIT (Cusumano & Yoffie, 1998). The judo strategy consisted of four broad factors: (a) scaling an organization on Internet time, (b) formulating a judo strategy on Internet time, (c) designing software on Internet time, and (d) developing software on Internet time. The last two factors comprised Netscape's software process: (a) designing software on Internet time and (b) developing software on Internet time. Designing software on Internet time consisted of four broad factors: (a) design products for multiple markets (platforms) concurrently, (b) design and redesign products to have more modular architectures, (c) design common components that multiple product teams can share, and (d) design new products and features for parallel development. Developing software on Internet time consisted of four broad factors: (a) adapt development priorities as products, markets, and customers change; (b) allow features to evolve but with frequent synchronizations and periodic stabilizations; (c) automate as much testing as possible; and (d) use beta testing, internal product usage, and other measures to improve product and process quality. Four less obvious software practices were embedded within Netscape's judo strategy: (a) solicit early market feedback from beta releases, (b) produce beta versions from which to solicit early market feedback, (c) form small decentralized teams, and (d) develop flexible cross-platform modularized software architectures. Several factors were attributed to Netscape's early success: (a) first mover status in new markets such as browsers, (b) strategic alliances that allowed it to exhibit a formidable market presence, (c) its vision to pioneer new and emerging market niches, and, more importantly, (d) its ability to develop new products at previously unheard of speeds.

Extreme programming. Extreme programming or XP was a software development method created by consultants designing a payroll system for Chrysler (Anderson et al., 1998). XP was inspired by Internet time and extreme sports as symbols of risk, speed, and danger, to build brand equity in the products and services of computer firms (Khermouch & Voight, 1997). Unimpressed by ethereal factors of Internet time, XP was based on more tangible and traditional techniques dating back to the earliest stages of the computer and software industries. Some of these foundational techniques included pair programming, joint application design, rapid application development, and the dynamic systems development method (Beck, 1999). XP took something unique from Internet time that was not found in traditional techniques. Based on Internet time, XP required its users to complete a development cycle and produce operational software every 14 days, unlike traditional methods and techniques (Beck, 1999). Originally, XP was based on four major factors: (a) simplicity, (b) customer-developer communication, (c) testing, and (d) aggressiveness (Anderson et al., 1998). The next version of XP included 13 major factors: (a) planning game, (b) small releases, (c) metaphor, (d) simple design, (e) tests, (f) refactoring, (g) pair programming, (h) continuous integration, (i) collective ownership, (j) on-site customer, (k) 40-hour weeks, (l) open workspace, and (m) just rules. The latest version of XP has 28 factors: user stories, release planning, frequent small releases, measuring velocity, iterations, iteration planning, personnel mobility, standup meetings, process improvement, simplicity, system metaphors, class-responsibility-collaboration cards, spike solutions, simplicity, refactoring, available customers, coding standards, automated unit tests, pair programming, sequential integration, frequent integration, collective ownership, late optimization, no overtime, complete unit tests, complete unit testing, debugging, and acceptance testing (Extreme Programming, 2006). Provocatively titled, XP became one of the widest used software methods.

Feature driven development. In 1996, Jeff De Luca created the "feature driven development" or FDD method to turnaround a failed banking project (Palmer & Felsing, 2002). Representing best known practices, FDD was modeled after joint application design, Coad's object oriented method, iterative development, code inspections, and chief programmer teams (Palmer & Felsing, 2002). FDD consisted of five processes: (a) develop an overall model, (b) build a features list, (c) plan by feature, (d) design by feature, and (e) build by feature (Coad, Lefebre, & De Luca, 1999). Originally, FDD only had three roles: (a) chief programmer, (b) class owner, and (c) feature team (Coad, Lefebre, & De Luca, 1999). However, FDD evolved to having six major roles: (a) project manager; (b) chief architect; (c) development manager; (d) chief programmer; (e) class owner; and (f) domain experts such as users, clients, sponsors, and analysts (Palmer & Felsing, 2002). Early customer involvement occurred in acceptance testing and three FDD processes: (a) develop an overall model, (b) build a features list, and (c) design by feature (Palmer & Felsing, 2002). Design and build iterations or iterative design occurred in FDD's last two processes (a) design by feature and (b) build by feature (Palmer & Felsing, 2002). Small teams were formed in each of FDD's first four processes: (a) develop an overall model, (b) build a features list, (c) plan by feature, and (d) design by feature (Palmer & Felsing, 2002). Evolution of its technical architectures generally occurred within the first four iterations of FDD's five major processes (Palmer & Felsing, 2002). That is, the development of complex architectures may be the highest priority. FDD also consisted of eight major practices: (a) domain object modeling, (b) developing by feature, (c) class (code) ownership, (d) feature teams, (e) inspections, (f) regular build schedule, (g) configuration management, and (h) reporting/visibility of results. Today, FDD has a small, but noticeable market as compared to extreme programming, scrum, and open source software development.

Open source software development. The term “open source software” or OSS was coined in 1997, though the practice of open source software started in 1970 (Bretthauer, 2002). Simply put, open source software was a “set of computer instructions that may be used, copied, modified, and distributed by anyone, anywhere, and for any purpose whatsoever” (Fink, 2003). Another definition stated “open source development is labeled with free source, fast evolution, and extensive user collaboration” (Zhao & Deek, 2004). One study identified eight major factors of OSS development: (a) is parallel rather than linear; (b) involves large communities of globally distributed developers; (c) utilizes truly independent peer review; (d) provides prompt feedback to user and developer contributions; (e) includes the participation of highly talented developers; (f) includes increased user involvement; (g) makes use of extremely rapid release schedules, and (h) produces evolutionary designs (Feller & Fitzgerald, 2002). Another early study identified the unique factors of OSS development for 11 communities of practice, such as: (a) Apache, (b) Gnome, (c) GCC, (d) Jakarta, (e) KDE, (f) Linux, (g) Mozilla, (h) NetBeans, (i) Perl, (j) Python, and (k) XFree86 (Halloran & Scherlis, 2002). A more recent study identified 15 factors of OSS development: (a) no forking a project, (b) no distribution without permission, (c) no removal of someone’s name from source code, (d) open source development methods produce better code, (e) outcomes are better when code is freely available, (f) outcomes are better when information is freely available, (g) more people will quickly find and fix bugs, (h) practical work is more useful than theory, (i) status is achieved through community, (j) sharing information is important, (k) aiding others is important, (l) technical knowledge is highly valued, (m) there is a value in learning, (n) voluntary cooperation is important, and (o) reputation is valuable (Stewart & Gosain, 2006). One author mused, “Internet time refers to something much faster, revolutionary, and more basic — It describes the process of developing open source software” (Pavlicek, 2000).

Agile manifesto. In 2001, the "agile manifesto" was created to outline the values and principles of agile methods and how they differed from traditional ones (Agile Manifesto, 2001). A council of 17 experts in agile methods met in order to find an "alternative to documentation-driven, heavyweight software development processes." They believed that agile methods rose to "free the developer community from the baggage of Dilbertesque corporations." Furthermore, they exclaimed "in order to succeed in the new economy, to move aggressively into the era of e-business, e-commerce, and the web, companies have to rid themselves of their Dilbert manifestations of make-work and arcane policies." Once the ground rules and assumptions of agile methods were established, they were able to get on with the business of writing the agile manifesto itself and publish it on the Internet. The agile manifesto began with the following statement: "we are uncovering better ways of developing software by doing it and helping others do it." Then the agile manifesto laid out four broad values: (a) "working software over comprehensive documentation," (b) "customer collaboration over contract negotiation," (c) "individuals and interactions over processes and tools," and (d) "responding to change over following a plan." The values of agile methods were capped off with the following statement, "while there is value in the items on the right, we value the items on the left more." In other words, they valued working software, customer collaboration, individuals and interactions, and responding to change much more than the comprehensive documentation, contract negotiation, processes and tools, and following a plan associated with traditional methods. They also devised 12 broad principles shown in Table 5. While there is an implicit semantic relationship between the values and principles found in the agile manifesto, there is no explicit mapping between the two as shown in Table 5. The principles were artificially arranged with the principles and factors for analytical purposes. That is, to understand the major factors of the agile manifesto.

Table 5. Analysis of Values and Principles of the Agile Manifesto

Values	Principles	Factors
Working software over comprehensive documentation	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software	**Iterative development**
	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale	
	Working software is the primary measure of progress	
Customer collaboration over contract negotiation	Business people and developers must work together daily throughout the project	**Customer feedback**
	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely	
	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly	
Individuals and interactions over processes and tools	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done	**Well-structured teams**
	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation	
	The best architectures, requirements, and designs emerge from self-organizing teams	
Responding to change over following a plan	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage	**Flexibility**
	Continuous attention to technical excellence and good design enhances agility	
	Simplicity--the art of maximizing the amount of work not done--is essential	

Subfactors of Iterative Development

Iterative development was defined as "an approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence" (Larman, 2004). Furthermore, "each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test." Iterative development is a software lifecycle, which is used to evolve operational software into finished products by incorporating customer feedback into its design. Iterative development is time-boxed, not scope-boxed, meaning that delivery dates are fixed by reducing product requirements. Sometimes iterative development is a scaled-back traditional software life cycle completed within three to six months (e.g., new development rhythm, dynamic systems development method, synch-n-stabilize, extreme programming, scrum, and feature driven development). In simpler, leaner approaches, iterative development refers to a much more dynamic daily release of beta versions to the market using the Internet (e.g., Internet time, judo strategy, and open source software development).

Table 6. Subfactors of Iterative Development

Method	Subfactors
New development rhythm	Iterative or cyclic process, developing system by iterating, functional milestones at each iteration, prototyping, staged delivery, overlapped component testing
Dynamic systems development	Frequent delivery of products, product-based approach, time-boxing, fixed end dates
Synch-n-stabilize	Risk-driven incremental spiral life cycle model, incremental milestones, prototypes, subprojects, daily builds, beta testing
Internet time	Evolutionary delivery, iterative approach, working version, prototypes, beta versions
Judo strategy	Short development cycles, three-month windows, multiple milestones, daily builds, internal usage testing, alpha testing, beta testing, field testing
Extreme programming	Release planning, release plans, iterations, iteration plans, frequent small releases, continuous integration, incremental deployment, daily deployment, incremental design
Scrum	Sprint planning meeting, product backlog, sprints, sprint backlog, sprint review meeting, sprint retrospective meeting, time-box, increment, shippable product, 30-day iteration
Feature driven development	Frequent deliveries, tangible working results, adaptive processes, feature development, small features, regular builds, feature lists, feature sets, feature designs, feature builds
Open source software	Rapid releases, rapid increments, multiple daily releases, development releases, production releases, early releases, official releases, new releases, minor releases, major releases

The new development rhythm consisted of six subfactors for iterative development: (a) iterative or cyclic process, (b) developing system by iterating, (c) functional milestones at each iteration, (d) prototyping, (e) staged delivery, and (f) overlapped component testing (Sulack, Lindner, & Dietz, 1989). The dynamic systems development method consisted of four subfactors of iterative development: (a) frequent delivery of products, (b) product-based approach, (c) time-boxing, and (d) fixed end dates (DSDM, 2007). Synch-n-stabilize consisted of six subfactors of iterative development: (a) risk-driven incremental spiral life cycle model, (b) incremental milestones, (c) prototypes, (d) subprojects, (e) daily builds, and (f) beta testing (Cusumano & Selby, 1995). Internet time consisted of five subfactors of iterative development: (a) evolutionary delivery, (b) iterative approach, (c) working version, (d) prototypes, and (e) beta versions (MacCormack, 2001). The judo strategy consisted of eight subfactors for iterative development: (a) short development cycles, (b) three-month windows, (c) multiple milestones, (d) daily builds, (e) internal usage testing, (f) alpha testing, (g) beta testing, and (h) field testing (Cusumano & Yoffie, 1998). Extreme programming consisted of nine subfactors of iterative development: (a) release planning, (b) release plans, (c) iterations, (d) iteration plans, (e) frequent small releases, (f) continuous integration, (g) incremental deployment, (h) daily deployment, (i) incremental design (Beck, 2005; Extreme Programming, 2006). Scrum consisted of 10 subfactors of iterative development: (a) sprint planning meeting, (b) product backlog, (c) sprints, (d) sprint backlog, (e) sprint review meeting, (f) sprint retrospective meeting, (g) time-box, (h) increment, (i) shippable product, and (j) 30-day iteration (Schwaber, 2004). Feature driven development consisted of 10 subfactors of iterative development: (a) frequent deliveries, (b) tangible working results, (c) adaptive processes, (d) feature development, (e) small features, (f) regular builds, (g) feature lists, (h) feature sets, (i) feature designs, and (j) feature builds (Palmer & Felsing, 2002). Open

source software development consisted of 10 subfactors for iterative development: (a) rapid release schedule, (b) rapid incremental releases, (c) multiple daily releases, (d) development releases, (e) production releases, (f) early releases, (g) official releases, (h) new releases, (i) minor releases, and (j) major releases (Halloran & Scherlis, 2002; Jorgensen, 2001).

Subfactors of Customer Feedback

Customer feedback is "a general term describing direct contact with users and covering many approaches" and lays on the "continuum from informative, through consultative to participative" (Kujala, 2003). Customer feedback has its origins from the earliest days when the discipline of marketing first emerged in the 1950s. Customer feedback was an important part of the systems dynamics movement of the 1960. Customer feedback was an important element of the customer active participation paradigm from the 1970s. Customer feedback was also an important part of the double loop learning paradigm of the early 1980s. Today, customer feedback is an essential element of most contemporary new product development approaches, especially ones that are fast, agile, flexible and able to respond to change. Succinctly stated, "customer feedback provides advantages in early identification of problems, effective screening of ideas, reducing design changes in later stages of development, and better defining the global market and opportunities" (Lim, Sharkey, & Heinrichs, 2003). Customer feedback is a means by which customers communicate their needs so that software developers may strive to fulfill them. Sometimes customer feedback refers to user participation in all life cycle activities (e.g., new development rhythm, dynamic systems development method, extreme programming, feature driven development, and open source software development). In simpler and leaner approaches, customer feedback refers to solicitation, receipt, and incorporation of market feedback on beta releases (e.g., synch-n-stabilize, Internet time, judo strategy, and scrum).

Table 7. Subfactors of Customer Feedback

Method	Subfactors
New development rhythm	Requirements briefings, field partners, usability activities, contract testing, migrations, invitational's, advisory councils, early support program
Dynamic systems development	Vision, business processes, requirements, designs, reviews, conflict management, measurement, monitoring, commitments, information, prototyping, approval, testing
Synch-n-stabilize	Planning data, wish lines, call data, usability, beta, and supportability testing, technical support, teleconferences, surveys, usage studies, instrumented tools, marketing studies
Internet time	Technical feedback on prototypes, technical feedback on daily operational builds, market feedback on early beta releases
Judo strategy	Technical feedback on alpha tests, internal feedback on beta tests, market feedback on beta tests, customer feedback from telephone support
Extreme programming	Integrated into team, provide feedback at all stages, write user stories, select user stories, prioritize user stories, specify test scenarios, approve tests, evaluate releases
Scrum	Attend reviews, ask questions, note changes, vote on impressions, changes, and priorities, rearrange backlogs, give feedback, identify omissions, suggest additions, add to backlog
Feature driven development	Participate in modeling team, give an overview of domain, assess domain model, help build features list, assess features list, participate in domain walkthrough
Open source software	Propose changes, vote on changes, report bugs, join mailing list, suggest guidelines, browse code, download code, analyze code, modify code, add code, join community

The new development rhythm consisted of eight subfactors for customer feedback: (a) requirements briefings, (b) field partners, (c) usability activities, (d) contract testing, (e) migrations, (f) migration invitational, (g) advisory councils, and (h) early support program (Pine, 1989). The dynamic systems development method had 13 subfactors for customer feedback, which consisted of ambassador and advisor users who assisted with: (a) visioning, (b) business processes, (c) requirements, (d) designs, (e) reviews, (f) conflict management, (g) measurement, (h) monitoring, (i) commitments, (j) information, (k) prototyping, (l) approval, and (m) testing (DSDM, 2007). The synch-n-stabilize method consisted of 12 subfactors of customer feedback: (a) planning data, (b) wish lines, (c) call data, (d) usability testing, (e) beta testing, (f) supportability testing, (g) technical support, (h) teleconferences, (i) surveys, (j) usage studies, (k) instrumented tools, and (l) marketing studies (Cusumano & Selby, 1995). The Internet time method consisted of three major subfactors of customer feedback: (a) technical feedback on prototypes, (b) technical feedback on daily operational builds, and (c) market feedback on early

beta releases (MacCormack, Verganti, & Iansiti, 2001). The judo strategy consisted of four major subfactors of customer feedback: (a) technical feedback on alpha tests, (b) internal feedback on beta tests, (c) market feedback on beta tests, and (d) customer feedback from telephone support (Cusumano & Yoffie, 1998). Extreme programming had eight subfactors of customer feedback, which consisted of customers who: (a) are integrated into teams, (b) provide feedback at all stages, (c) write user stories, (d) select user stories, (e) prioritize user stories, (f) specify test scenarios, (g) approve tests, and (h) evaluate releases (Extreme Programming, 2006). Scrum had nine subfactors of customer feedback, which consisted of stakeholders who: (a) attend reviews; (b) ask questions; (c) note changes; (d) vote on impressions, changes, and priorities; (e) rearrange backlogs; (f) give feedback; (g) identify omissions; (h) suggest additions; and (i) add to backlog (Schwaber, 2004). Feature driven development had six subfactors of customer feedback, which consisted of domain experts who: (a) participate in modeling team, (b) give an overview of domain, (c) assess domain model, (d) help build features list, (e) assess features list, and (f) participate in domain walkthrough (Palmer & Felsing, 2002). Open source software development had 11 subfactors of customer feedback: (a) propose changes, (b) vote on changes, (c) report bugs, (d) join mailing list, (e) suggest guidelines, (f) browse code, (g) download code, (h) analyze code, (i) modify code, (j) add code, and (k) join community (Halloran & Scherlis, 2002).

Subfactors of Well-Structured Teams

Well-structured teams are defined as “work groups who are made up of individuals who see themselves as a social entity, who are interdependent because of the tasks they perform, who are embedded in one or more larger social systems, and who perform tasks that affect others” (Guzzo & Dickson, 1996). Within the context of agile methods, well-structured teams are

defined as “groups who are responsible for setting direction, establishing boundaries, assigning staff with certain talents to roles, and building a multi-tiered decision-making process in which managers have the responsibility and authority to make certain decisions” (Highsmith, 2002). Within agile methods, the team has a clear leader, the leader is most likely a product line manager or a project manager, the leader has the resources and authority to organize the project, and the leader has authority over programmers. This definition for well-structured teams holds true for the new development rhythm, dynamic systems development, synch-n-stabilize, judo strategy, scrum, and feature driven development methods. However, these structures seem less apparent in extreme programming and open source software development. It is interesting to note, that in spite of traditional management structures, software engineers were able to put out daily releases of software using synch-n-stabilize, Internet time, and judo strategy, when needed. Even open source software developers have a clear leader, if not formalized project structures.

Table 8. Subfactors of Well-Structured Teams

Method	Subfactors
New development rhythm	Cross-functional teams, empowered management teams, isolated development teams, co-located teams, special decision teams, engineering teams, testing teams, design groups
Dynamic systems development	Team leaders, empowerment, large team structures, collaboration, communication, daily sub-team meetings, daily time-box meetings, core teams, facilitated workshops
Synch-n-stabilize	Small teams, overlapping functional specialists, delegated hiring, learning by doing, mentoring, career paths, ladder levels, specialized management and teams
Internet time	Small teams, broad-based experienced team, team empowered to respond to market feedback, team with generational experience
Judo strategy	Numerous teams, small six-person teams, decentralized teams, self-managed teams, product teams, programming teams, build teams, version teams, Unix teams, quality assurance teams
Extreme programming	Pair programming, personnel rotation, cross training, co-location, side-by-side, take breaks, humility, confidence, communication, listening, teamwork
Scrum	Self-managing teams, self-organizing teams, cross-functional teams, collective responsibility, daily scrum meetings
Feature driven development	Feature teams, team leaders, class owners, code inspection team, small teams, modeling teams, planning teams, development teams, feature list teams
Open source software	International communities, distributed communities, large communities, mailing lists, quality assurance groups, core teams, trust

The new development rhythm consisted of eight subfactors for well-structured teams: (a) cross-functional teams, (b) empowered management teams, (c) isolated development teams, (d) co-located teams, (e) special decision teams, (f) engineering teams, (g) testing teams, and (h) design groups (Sulack, Lindner, & Dietz, 1989). The dynamic systems development method consisted of nine subfactors for well-structured teams: (a) team leaders, (b) empowerment, (c) large team structures, (d) collaboration, (e) communication, (f) daily sub-team meetings, (g) daily time-box meetings, (h) core teams, and (i) facilitated workshops (DSDM, 2007). Synch-n-stablize consisted of eight subfactors for well-structured: (a) small teams, (b) overlapping functional specialists, (c) delegated hiring, (d) learning by doing, (e) mentoring, (f) career paths, (g) ladder levels, and (h) specialized management and teams (Cusumano & Selby, 1995). Internet time consisted of four subfactors for well-structured teams: (a) small teams, (b) broad-based experienced team, (c) team empowered to respond to market feedback, and (d) team with generational experience (MacCormack, Verganti, & Iansiti, 2001). The judo strategy consisted of 10 subfactors for well-structured teams: (a) numerous teams, (b) small six-person teams, (c) decentralized teams, (d) self-managed teams, (e) product teams, (f) programming teams, (g) build teams, (h) version teams, (i) Unix teams, and (j) quality assurance teams (Cusumano & Yoffie, 1998; Yoffie & Cusumano, 1999). Extreme programming consisted of 11 subfactors of well-structured teams: (a) pair programming, (b) personnel rotation, (c) cross training, (d) co-location, (e) side-by-side, (f) take breaks, (g) humility, (h) confidence, (i) communication, (j) listening, and (k) teamwork (Extreme Programming, 2006; Williams & Kessler, 2003). Scrum consisted of five subfactors of well-structured teams: (a) self-managing teams, (b) self-organizing teams, (c) cross-functional teams, (d) collective responsibility, and (e) daily scrum meetings (Schwaber, 2004). Feature driven development consisted of nine subfactors of well-

structured teams: (a) feature teams, (b) team leaders, (c) class owners, (d) code inspection team, (e) small teams, (f) modeling teams, (g) planning teams, (h) development teams, and (i) feature list teams (Palmer & Felsing, 2002). Open source software development consisted of seven subfactors of well-structured teams: (a) international communities, (b) distributed communities, (c) large communities, (d) mailing lists, (e) quality assurance groups, (f) core teams, and (g) trust (Halloran & Scherlis, 2002).

Subfactors of Flexibility

Flexibility is defined as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed" (Institute of Electrical and Electronics Engineers, 1990). This is a highly relevant definition of flexibility because the goal of agile methods is to form an organizational structure in which to evolve software designs until they satisfy their customer's needs. Therefore, software designs themselves must be created in such a way that they can be modified over and over again. Three closely related definitions worth mentioning include: (a) "a flexible product development process is one that allows designers to continue to define and shape products even after implementation has begun" (Iansiti & MacCormack, 1997); (b) "a flexible process is characterized by the ability to generate and respond to new information for a longer proportion of a development cycle" (MacCormack, Verganti, & Iansiti, 2001); and (c) "development flexibility can be expressed as a function of the incremental economic cost of modifying a product as a response to changes that are external or internal to the development process" (Thomke & Reinertsen, 1998). All of these definitions allude to both a new product development process and, more specifically, a product architecture, which in combination allow for a product to gradually grow, expand, and evolve as engineers solicit, acquire, and incorporate more and more end user, customer, and market needs.

It is important to note that the creators of agile methods (e.g., Highsmith, 2002) prefer to link agile methods to early theories of adaptive organizations (e.g., Callahan, 1979). However, definitions of flexibility found in theories of Internet time (Iansiti & MacCormack, 1997) are better suited to describe the factors of agile methods, because of their scholarly nature. All of the subfactors for the agile methods in Table 9 adhered to the four definitions of flexibility offered here (e.g., the ease of which software can be modified, a process that allows designers to continuously evolve products, processes with the ability to respond to new information, and designs that cost-effectively accommodate changes). While all of the agile methods in Table 9 simply imply projects must spend resources creating software architectures that accommodate change, very few say exactly how to create a flexible architecture. The judo strategy alludes to cross-platform designs (e.g., software that runs on as many computers as possible). However, technologies such as Java have substantially matured since the judo strategy was formed, which enable firms using all of these agile methods to create cross-platform software products.

Table 9. Subfactors of Flexibility

Method	Subfactors
New development rhythm	Preliminary system architectures, experimental system architectures, architectural reuse, software reuse, design control groups, early system, software, and user interface prototypes
Dynamic systems development	Fitness for business purpose, system architecture definition, enterprise model, system model, technology model, reversible changes
Synch-n-stabilize	Vision statements, evolving specifications, horizontal architectures, modularity, functional building blocks, flexible skeletons, architectural layers, portable designs, simple code
Internet time	Flexible, evolving, coherent, delayed, robust, open, scaleable, and modular architectures and designs
Judo strategy	Cross-platform modularized architectures, designs, programming systems, programming languages, feature sets, abstraction layers, components, and reusable libraries
Extreme programming	Architectural spikes, system metaphors, spikes, spike solutions, simple designs, delayed functionality, merciless refactoring, coding standards, delayed optimization
Scrum	Product infrastructures, detailed product architectures, detailed technical architectures, business architecture, system architecture, development environment
Feature driven development	Technical architectures, user interface layers, problem domain layers, data management layers, system interaction layers, domain object models, class diagrams, sequence diagrams
Open source software	Advanced design decisions, solid architectures, design patterns, portable designs, modular designs, code modularity, cohesive modules, coding guidelines, standards, and conventions

The new development rhythm consisted of six subfactors of flexibility: (a) preliminary system architectures; (b) experimental system architectures; (c) architectural reuse; (d) software reuse; (e) design control groups; and (f) early system, software, and user interface prototypes (Sulack, Lindner, & Dietz, 1989). The dynamic system development method consisted of six subfactors of flexibility: (a) fitness for business purpose, (b) system architecture definition, (c) enterprise model, (d) system model, (e) technology model, and (f) reversible changes (DSDM, 2007). Synch-n-stabilize consisted of nine subfactors of flexibility: (a) vision statements, (b) evolving specifications, (c) horizontal architectures, (d) modularity, (e) functional building blocks, (f) flexible skeletons, (g) architectural layers, (h) portable designs, and (i) simple code (Cusumano & Selby, 1995). Internet time consisted of eight subfactors of flexibility: (a) flexible, (b) evolving, (c) coherent, (d) delayed, (e) robust, (f) open, (g) scaleable, and (h) modular architectures and designs (Iansiti & MacCormack, 1997; MacCormack, Verganti, & Iansiti, 2001). The judo strategy had eight subfactors of flexibility, which consisted of cross-platform modularized: (a) architectures, (b) designs, (c) programming systems, (d) programming languages, (e) feature sets, (f) abstraction layers, (g) components, and (h) reusable libraries (Cusumano & Yoffie, 1998). Extreme programming consisted of nine subfactors of flexibility: (a) architectural spikes, (b) system metaphors, (c) spikes, (d) spike solutions, (e) simple designs, (f) delayed functionality, (g) merciless refactoring, (h) coding standards, and (i) delayed optimization (Extreme Programming, 2006). Scrum consisted of six subfactors of flexibility: (a) product infrastructures, (b) detailed product architectures, (c) detailed technical architectures, (d) business architecture, (e) system architecture, and (f) development environment (Schwaber, 2004). Feature driven development consisted of eight subfactors of flexibility: (a) technical architectures, (b) user interface layers, (c) problem domain layers, (d) data management layers,

(e) system interaction layers, (f) domain object models, (g) class diagrams, and (h) sequence diagrams (Palmer & Felsing, 2002). Open source software development consisted of eight subfactors of flexibility: (a) advanced design decisions; (b) solid architectures; (c) design patterns; (d) portable designs; (e) modular designs; (f) code modularity; (g) cohesive modules; and (h) coding guidelines, standards, and conventions (Feller & Fitzgerald, 2002; Halloran & Scherlis, 2002).

Hypotheses Linking Subfactors of Agile Methods to Website Quality

Iterative development. The underlying assumption for the last 30 years has been that subfactors of iterative development are linked to software development success factors, such as lower risks, lower system complexity, more accurate project estimates, fewer defects, user satisfaction, and many others (Larman, 2004). Subfactors of iterative development have been linked to better market performance (Li & Calantone, 1998), better product performance (Di Benedetto, 1999), and better firm performance (Jin, 2000). Subfactors of iterative development have been linked to faster cycle times (Sherman, Souder, & Jenssen, 2000), improved customer relationships (Stump, Athaide, & Joshi, 2002), and higher satisfaction with firm-level partnerships (Athaide, Stump, & Joshi, 2003). Subfactors of iterative development have also been linked to better project performance (Joshi & Sharma, 2004), fewer engineering hours (Hong, Vonderembse, Doll, & Nahm, 2005), and higher product quality (Schulze & Hoegl, 2006). More importantly, subfactors of iterative development have been linked to higher website quality (MacCormack, 2001). However, low numbers of iterations (e.g., less than six) have not been proven to be linked to improved website quality, though higher numbers of iterations have been linked to improved product performance (Allen, 1966; Thomke & Reinertsen, 1998). While there is some indication that factors of iterative development were linked to higher website

quality, the field of scholarly models of website quality has yet to mature. Thus, we will seek to link subfactors of iterative development to scholarly models of website quality:

Hypothesis 1 (H_1): Iterative development is linked to higher website quality

Customer feedback. The underlying assumption for nearly 50 years has been that subfactors of customer feedback are linked to software development success factors such as higher productivity, quality, performance, and even user satisfaction (Ives & Olson, 1984). Subfactors of customer feedback have been linked to better conflict management (Barki & Hartwick, 1994a), improved user attitudes (Barki & Hartwick, 1994b), and system use (Hartwick & Barki, 1994). Subfactors of customer feedback have also been linked to higher user satisfaction (McKeen, Guimaraes, & Wetherbe, 1994), higher user productivity (Hunton & Beeler, 1997), and better project performance (Jiang, Chen, & Klein, 2002). More to the point, subfactors of customer feedback have been linked to higher information quality (Blili, Raymond, & Rivard, 1998), higher system quality (Barki & Hartwick, 2001), and system performance (Rondeau, Ragu-Nathan, & Vonderemsbe, 2006). Even closer to home, subfactors of customer feedback have been linked to higher website quality (MacCormack, 2001). Furthermore, customer feedback must not only be sought and obtained, but must be acted upon in order to improve quality (MacCormack, 2001). While there is some indication that factors of customer feedback were linked to higher website quality, the field of scholarly models of website quality has yet to mature. Thus, we will seek to link subfactors of customer feedback to scholarly models of website quality:

Hypothesis 2 (H_2): Customer feedback is linked to higher website quality

Well-structured teams. The underlying assumption for the last 60 years has been that subfactors of well-structured teams have been linked to performance factors such as higher

morale, satisfaction, efficiency, capacity, productivity, pride in workmanship, and many others (Herbst, 1962). Subfactors of well-structured teams have been linked to better team performance (Guinan, Cooprider, & Faraj, 1998), greater group cohesiveness (Riordan & Weatherly, 1999), and better team effectiveness (Langfred, 2000). Subfactors of well-structured teams have been linked to personal work satisfaction (Hoegl & Gemuenden, 2001), improved teamwork (Paul, Samarah, Seetharaman, & Mykytyn, 2004), and more creativity (Tiwana & McLean, 2005). Subfactors of well-structured teams have been linked to new product success (Lynn, Skov, & Abel, 1999), system success (Sawyer, 2001), and faster time to market (Sarin & McDermott, 2003). In a recent study, subfactors of well-structured teams were linked to higher task completion and greater team effort among open source software development teams, which are a form of agile methods (Stewart & Gosain, 2006). Subfactors of well-structured teams such as task interdependence have been linked to better performance among software development teams (Edwards & Sridar, 2005). It is important to note that a major study of agile methods failed to link subfactors of well-structured teams, such as experience, to website quality (MacCormack, 2001). Thus, we will seek to link subfactors of well-structured teams, other than experience alone, to scholarly models of website quality:

Hypothesis 3 (H_3): Well-structured teams are linked to higher website quality

Flexibility. The underlying assumption for the last 50 years has been that subfactors of flexibility have been linked to software success factors such as quality, reliability, maintainability, and cost effectiveness (Parnas, 1972). Subfactors of flexibility have been linked to improved competitiveness (Byrd & Turner, 2001), better system performance (Nelson & Cooprider, 2001), and better organizational performance (Chung, Rainer, & Lewis, 2003). Subfactors of flexibility have been linked to organizational agility (Palanisamy & Sushil, 2003),

better system efficiency (Golden & Powell, 2004), and faster time to market (Singh & Sushil, 2004). Subfactors of flexibility have been linked to improved partnerships (Gosain, Mulhotra, & El Sawy, 2005), more frequent project success (Xia & Lee, 2005), and an organization's return on sales (Zhang, 2005). Closely related to this study, subfactors of flexibility among Internet firms have been linked to improved website quality as well as organizational performance (Cusumano & Selby, 1995; Iansiti & MacCormack, 1997; Yoffie & Cusumano, 1999). In at least two of these studies, subfactors of flexibility were strong predictors of website quality and organizational performance, but neither linked the subfactors of flexibility to scholarly models of website quality. Thus, we will seek to link subfactors of flexibility to scholarly models of website quality:

Hypothesis 4 (H_4): Flexibility is linked to higher website quality

Conceptual Framework Summary

There have been hundreds of studies of software methods since the 1950s, there have been hundreds of studies on what constitutes system success, and there have been hundreds of studies linking subfactors of software methods to the subfactors of system success. There have even been some rather intriguing studies of agile methods among Internet firms, subfactors of agile methods among Internet firms, and the impacts of agile methods on the organizational performance of Internet firms. However, there are several limitations associated with some of these more recent studies of agile methods among Internet firms: (a) none of the better studies have a theoretical conceptual model, (b) few of them were based on all four major factors of agile methods, and (c) none of them attempted to link the factors of agile methods to scholarly models of website quality. Thus, the conceptual framework (e.g., model, factors, subfactors, and hypotheses) identified here may be one of the first holistic theories of agile methods.

RESEARCH METHOD

The purpose of this section is to present a research method for collecting measurements, which can then be used to analyze the relationships between agile methods and website quality. The research method must help determine whether: (a) agile methods are being used, (b) high quality websites are being produced, and (c) the use of agile methods is linked to website quality. First, a survey will be conducted to determine the degree to which software developers are using agile methods. Second, an independent assessment of the quality of their websites will be performed. Third, the relationships between the use of agile methods and website quality will be analyzed and studied. The main survey instrument is based on the four major factors of agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. The instrument to measure website quality is also composed of four factors: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service.

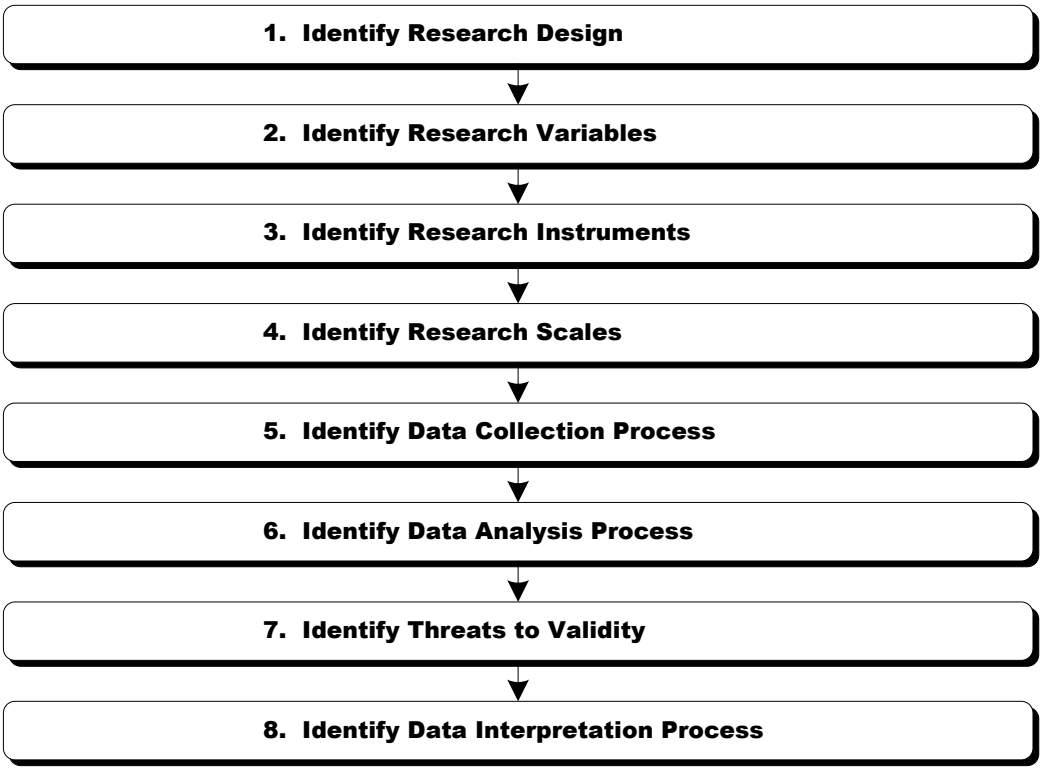

Figure 8. Research method for studying agile methods and website quality.

Research Design

Survey method. The recommended research design is a small-scale survey in order to measure the relationships between the four factors of agile methods and scholarly models of website quality. Survey research is one of the premier approaches for collecting both small and large quantities of data to support scholarly research in the fields of both administrative and management science. Survey research has been successfully used to study the four factors of agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. Survey research is the preferred research design used by management scientists to collect and analyze attitudinal data concerning use of the Internet and website quality. The goal of this study is to help determine whether a link exists between the use of agile methods and website quality. Therefore, survey research will help satisfy this goal by gathering data about the quality of websites produced using agile methods.

Population and sample. The population of computer programmers in the U.S. is around 2 million and has been in steady decline since 2001. The level of U.S. electronic commerce revenues has grown to more than $2 trillion, in spite of the decline in both U.S. information technology workers and computer programmers. The sample cannot equal the population for a full-scale academic research study and certainly not for a small-scale graduate study. The sample will consist of up to 30,000 subscribers to a major U.S. computer programming journal. The subscribers largely represent managers and programmers within the U.S. computer programming industry. Recent response rates to similar surveys using this journal have ranged from 700 (2.3%) to 2,500 (8.3%) respondents. So, we can expect around 300 (1%) of all of the magazine's subscribers to respond to our survey of agile methods and website quality. This is more than enough for small-scale graduate survey research such as this.

Research Variables

Agile methods. The main survey instrument is designed to measure compliance with agile methods and consists of four major factors: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. Iterative development consists of five variables: (a) time boxed releases, (b) operational releases, (c) small releases, (d) frequent releases, and (e) numerous releases. Customer feedback consists of five variables: (a) feedback solicited, (b) feedback received, (c) feedback frequency, (d) feedback quality, and (e) feedback incorporated. Well-structured teams consists of five variables: (a) team leader, (b) vision and strategy, (c) goals and objectives, (d) schedules and timelines, and (e) small team size. Flexibility consists of five variables: (a) small size, (b) simple design, (c) modular design, (d) portable design, and (e) extensible design. The survey instrument in Table 10 contains further definitions and explanations of the 20 variables used to measure compliance with agile methods.

Website quality. The secondary survey instrument is designed to assess the quality of e-commerce websites produced using agile methods and also consists of four major factors: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. Website design consists of five variables: (a) in-depth information, (b) processing efficiency, (c) processing speed, (d) personalization, and (e) product selection. Privacy and security consists of three variables: (a) protection of privacy, (b) feelings of safety, and (c) adequate security. Fulfillment and reliability consists of three variables: (a) order received, (b) on time delivery, and (c) order accurate. Customer service consists of three variables: (a) willingness to respond, (b) desire to fix issues, and (c) promptness of service. The survey instrument in Table 11 contains definitions and explanations of the 14 variables used to measure e-commerce website quality, which is based upon eTailQ (Wolfinbarger & Gilly, 2003).

Research Instruments

Agile methods. The survey instrument for agile methods was designed using the factors and subfactors associated with the conceptual model of agile methods (as shown in Table 10). The major factors of agile methods were derived from an analysis of scholarly literature on agile methods. Four major factors are associated with agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. Similarly, the subfactors of agile methods were derived from an analysis of the same scholarly literature of agile methods. The items (i.e., questions) were carefully phrased based on the intent of the factors and subfactors as they were used in the scholarly literature from which they were derived.

Table 10. Measurement Instrument for Assessing Agile Methods

Factor	Variable	Item
Iterative development	Time boxed releases	We develop software using time-based iterations, increments, or demonstrations
	Operational releases	We develop software using operational iterations, increments, or demonstrations (working code)
	Small releases	We develop software using small iterations, increments, or demonstrations
	Frequent releases	We develop software using daily, weekly, bi-weekly, or monthly iterations, increments, or demonstrations
	Numerous releases	We develop software using multiple (several) iterations, increments, or demonstrations
Customer feedback	Feedback solicited	We seek customer feedback on our software iterations, increments, or demonstrations
	Feedback received	We receive customer feedback on our software iterations, increments, or demonstrations
	Feedback frequency	We receive timely customer feedback on our software iterations, increments, or demonstrations
	Feedback quality	We receive a lot of (detailed) customer feedback on our software iterations, increments, or demonstrations
	Feedback incorporated	We incorporate customer feedback into our software iterations, increments, or demonstrations
Well-structured teams	Team leader	Our software teams have clear administrative or technical leaders
	Vision and strategy	Our software teams have clear visions, missions, or strategies
	Goals and objectives	Our software teams have clear goals or objectives
	Schedules and timelines	Our software teams have clear schedules or timelines
	Small team size	Our software teams have a small size with no more than 10 people
Flexibility	Small size	Our software is designed to be as small as possible
	Simple design	Our software is designed to be as simple as possible
	Modular design	Our software is designed to be modular or object-oriented
	Portable design	Our software is designed to work on multiple operating systems
	Extensible design	Our software is designed to be changed, modified, or maintained

Website quality. As shown in Table 11, the survey instrument for website quality was derived from the factors, subfactors, and questions from eTailQ (Wolfinbarger & Gilly, 2003). The eTailQ instrument itself was designed from an analysis of scholarly literature, use of focus groups, and extensive field testing and validation of data collected from over 1,000 respondents. Four major factors are associated with eTailQ: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. Furthermore, only 14 instrument items are associated with the eTailQ instrument: (a) five for website design, (b) three for protection and privacy, (c) three for fulfillment and reliability, and (d) three for customer service. The strengths of eTailQ are: (a) it is one of the newest, most mature, and scholarly models of website quality; (b) it encompasses the entire life cycle associated with electronic commerce transactions; (c) it has a small size maximizing speed and clarity; (d) it was developed and tested using focus groups, confirmatory factor analysis, and a large population of respondents; and (e) each of its factors, variables, and items exhibit high levels of inter-item reliability and validity.

Table 11. Measurement Instrument for Assessing Website Quality

Factor	Variable	Item
Website design	In-depth information	The website provides in-depth information
	Processing efficiency	The site doesn't waste my time
	Processing speed	It is quick and easy to complete a transaction at this website
	Personalization	The level of personalization at site is about right, not too much or too little
	Product selection	This website has good selection
Privacy and security	Protection of privacy	I feel like my privacy is protected at this site
	Feelings of safety	I feel safe in my transactions with this website
	Adequate security	The website has adequate security features
Fulfillment and reliability	Order received	You get what you ordered from this site
	On time delivery	The product is delivered by the time promised by the company
	Order accurate	The product that came was represented accurately by the website
Customer service	Willingness to respond	The company is willing and ready to respond to customer needs
	Desire to fix issues	When you have a problem, the website shows a sincere interest in solving it
	Promptness of service	Inquiries are answered promptly

Research Scales

All of the instrument items for the survey instruments will consist of a seven point Likert-type scale. They will be rank-ordered from lowest to highest as follows: (a) strongly disagree, (b) disagree, (c) somewhat disagree, (d) neutral, (e) somewhat agree, (f) agree, and (g) strongly agree. A graduated Likert-type scale will maximize the degree of variability in the responses and lend itself to an analysis of continuous variable data. Rank-ordering the responses from lowest to highest will also reduce the likelihood of misclassifying or incorrectly categorizing the data due to reverse encoding of response data. For instance, the scales will be encoded from one to seven, depending upon the response (e.g., 1 for strongly disagree and 7 for strongly agree). Each of the instrument items was designed to elicit a response corresponding to these scales. That is, items were designed to elicit a graduated response, rather than simply “yes” or “no.”

Data Collection Process

The goal of this study is to test for a link between the use of agile methods and website quality. We will accomplish this goal by assessing the quality of websites produced by computer programmers using agile methods. In order to accomplish the goals of this study, we will first conduct up to six cognitive interviews to further validate the agile methods survey instrument. We will then survey up to 300 software developers using the agile methods survey instrument to determine the extent to which they are using agile methods. We will also ask these developers to provide the addresses of websites they have produced using agile methods. From there, we will randomly choose which website addresses to use. Then we will conduct an independent assessment of the quality of the websites. In order to gauge the progress of the data collection process, we will also collect interim, self-report data on the benefits of agile methods, such as improvements in cost efficiency, productivity, quality, cycle time, and customer satisfaction.

Data Analysis Process

The first step is to obtain a usable data set from the questionnaires. The next step is to generate a summary of descriptive statistics. We propose to do a simple statistical analysis of the responses from the sample of software developers. We will do this by analyzing the response data from the instrument items, variables, and factors from the agile methods and eTailQ survey instruments. Thus, we will determine the degree to which the websites produced by programmers satisfy the variables and factors of agile methods and eTailQ. Then, we will attempt to analyze the relationships between the use of agile methods and the quality of e-commerce websites produced by U.S. programmers. We fully expect all of the websites produced by programmers using agile methods to meet or exceed the minimum thresholds of website quality using eTailQ. Correlational analysis and linear regression will be the primary statistical methods employed.

Threats to Validity

There are several major threats to the validity of this study. First, the instrument to measure agile methods is new and untested, so its reliability cannot be fully determined in advance. Second, the website quality instrument has only been tested on perceptions of past transactions involving e-commerce website quality. It has never been used to assess the quality of specific websites. Third, the respondents are self-selected, so there may be some bias towards the use of agile methods. This is not likely given the amount skepticism regarding the validity of agile methods. Fourth, we may not collect enough data about agile methods or website quality to yield significant relationships between agile methods and website quality. Fifth, survey research may or may not be the best research method to analyze the impacts of agile methods, in lieu of qualitative methods, which yield richer experiences. Sixth, linear regression may not be sensitive enough to measure minute variations in the data we collect.

Data Interpretation Process

The primary method of interpreting results of the data analysis will be hypothesis testing. Correlational analysis will be performed, linear regression between variables and factors will be performed, and statistical models will be built and tested for correlations between all of the major factors. Finally, statistical models will be designed and built to correlate all of the factors of agile methods to the factors of website quality, including a composite model of all of the factors of website quality. Models will be built between website quality and iterative development, customer feedback, well structured teams, and flexibility to test the hypotheses. Then individual models will be built between website design, privacy and security, fulfillment and reliability, and customer service and the factors of agile methods to test the sub-hypotheses. These results will not only test for large magnitudes of correlations, but statistically significant ones as well. Finally, summary tables will be populated with these results (as shown in Table 12).

Table 12. Method of Interpreting Results of Data Analysis Process

Factors	Hypothesis	β	*t-value*	*p-value*
Iterative development	H_1 Iterative development → Website quality	+/–	+/–	$p < 0.05$
	H_{1a} Iterative development → Website design	+/–	+/–	$p < 0.05$
	H_{1b} Iterative development → Privacy and security	+/–	+/–	$p < 0.05$
	H_{1c} Iterative development → Fulfillment and reliability	+/–	+/–	$p < 0.05$
	H_{1d} Iterative development → Customer service	+/–	+/–	$p < 0.05$
Customer feedback	H_2 Customer feedback → Website quality	+/–	+/–	$p < 0.05$
	H_{2a} Customer feedback → Website design	+/–	+/–	$p < 0.05$
	H_{2b} Customer feedback → Privacy and security	+/–	+/–	$p < 0.05$
	H_{2c} Customer feedback → Fulfillment and reliability	+/–	+/–	$p < 0.05$
	H_{2d} Customer feedback → Customer service	+/–	+/–	$p < 0.05$
Well-structured teams	H_3 Well-structured teams → Website quality	+/–	+/–	$p < 0.05$
	H_{3a} Well-structured teams → Website design	+/–	+/–	$p < 0.05$
	H_{3b} Well-structured teams → Privacy and security	+/–	+/–	$p < 0.05$
	H_{3c} Well-structured teams → Fulfillment and reliability	+/–	+/–	$p < 0.05$
	H_{3d} Well-structured teams → Customer service	+/–	+/–	$p < 0.05$
Flexibility	H_4 Flexibility → Website quality	+/–	+/–	$p < 0.05$
	H_{4a} Flexibility → Website design	+/–	+/–	$p < 0.05$
	H_{4b} Flexibility → Privacy and security	+/–	+/–	$p < 0.05$
	H_{4c} Flexibility → Fulfillment and reliability	+/–	+/–	$p < 0.05$
	H_{4d} Flexibility → Customer service	+/–	+/–	$p < 0.05$

DATA ANALYSIS

The purpose of this section is to present an analysis of the relationships between the use of agile methods to make e-commerce websites and the resulting website quality. A survey was used to capture information on three major groups of data: (a) the use of agile methods, (b) the benefits of using agile methods, and (c) the quality of the resulting websites. The data analysis process consists of analyzing descriptive, demographic, agile methods, benefit, and website quality data, in addition to the relationships between the three major groups of data. Statistical software was used to analyze the descriptive, demographic, agile methods, benefit, and website quality data, along with the relationships between these last three major groups of data. Approximately 250 respondents provided data on agile methods, 150 respondents provided data on benefits of agile methods, and 10 respondents provided addresses of e-commerce websites. Data analysis revealed correlations within groups of data, correlations between agile methods and benefit data, and insignificant relationships between agile methods and website quality data.

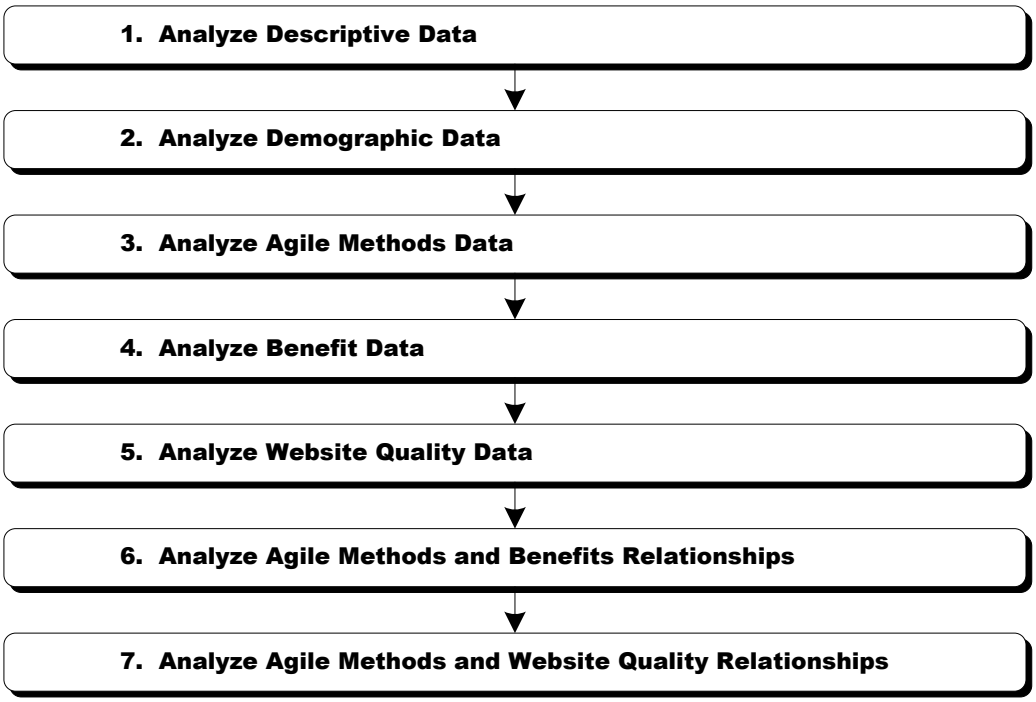

Figure 9. Data analysis of relationships between agile methods and website quality.

Descriptive Data

Data were collected for agile methods, project outcomes, and website quality. Almost 250 respondents provided data on agile methods, 150 respondents provided data on project outcomes, and 10 respondents provided data on website quality. The first two groups were self-reported data and the last group was based on an independent assessment of website quality.

Table 13. Descriptive Data on Agile Methods, Benefits, and Website Quality

Group	Factor	Variable	N	Min	Max	Mean	σ
AGILE METHODS	Iterative development	Time-boxed releases	249	1	7	5.15	1.91
		Operational releases	249	1	7	5.15	1.68
		Small releases	246	1	7	5.13	1.83
		Frequent releases	247	1	7	4.70	2.10
		Numerous releases	248	1	7	5.38	1.73
	Customer feedback	Feedback solicited	249	1	7	5.55	1.60
		Feedback received	249	1	7	5.24	1.57
		Feedback frequency	246	1	7	4.45	1.67
		Feedback quality	246	1	7	4.05	1.83
		Feedback incorporated	249	1	7	5.67	1.48
	Well-structured teams	Team leader	251	1	7	5.25	1.61
		Vision and strategy	250	1	7	4.82	1.63
		Goals and objectives	249	1	7	5.20	1.42
		Schedules and timelines	246	1	7	5.07	1.59
		Small team size	249	1	7	5.96	1.55
	Flexibility	Small size	249	1	7	4.22	1.70
		Simple design	249	1	7	4.82	1.71
		Modular design	248	1	7	5.38	1.50
		Portable design	248	1	7	4.08	2.09
		Extensible design	251	1	7	5.24	1.57
PROJECT OUTCOME	Benefits	Cost	122	1	7	3.07	1.80
		Productivity	145	1	7	3.32	1.73
		Quality	149	1	7	3.48	1.88
		Cycle time	138	1	7	3.55	1.91
		Customer satisfaction	136	1	7	3.77	1.91
WEBSITE QUALITY	Website design	In-depth information	10	5	7	6.10	0.57
		Processing efficiency	10	3	7	5.80	1.23
		Processing speed	10	2	7	5.00	2.05
		Personalization	10	3	7	5.60	1.43
		Product selection	10	2	7	5.80	1.40
	Privacy and security	Protection of privacy	10	4	7	6.00	1.41
		Feelings of safety	10	4	7	6.00	1.41
		Adequate security	10	2	7	5.50	1.84
	Fulfillment and reliability	Order received	10	1	7	5.20	1.99
		On time delivery	10	1	7	5.20	1.99
		Order accurate	10	1	7	5.20	1.99
	Customer service	Willingness to respond	10	1	7	5.70	1.95
		Desire to fix issues	10	1	7	5.70	1.95
		Promptness of service	10	1	7	5.70	1.95

There were an adequate number of data points on agile methods and project outcomes to analyze the relationships within and between these groups of variables. A total of 27 Internet addresses were provided, but only 10 of them were for e-commerce websites versus information websites. The data points represent values on a seven-point Likert type scale. The means were quite high for agile methods, which may mean that respondents agreed with statements about these variables. The means for project outcomes were lower, which means respondents were a little more conservative about their statements with regards to benefits. The means for website quality were higher, but the low number of Internet addresses would prove problematic for examining the relationships between agile methods and website quality. The latter could have been avoided by asking for self-reported data on website quality in addition to an Internet address of an e-commerce website for an independent assessment of its quality.

Demographic Data

Demographic data were collected for job function, years of experience, number of employees, annual revenues, and industry sector. The response rate for demographic data was quite high: (a) 250 respondents for job function, (b) 252 respondents for years of experience, (c) 245 respondents for number of employees, (d) 161 respondents for annual revenue, and (e) 230 respondents for industry sector. Software engineers represented the largest job function at 33%. However, 38% of the job functions ranged from executives to project managers. The largest group of respondents had 11 to 15 years of experience, came from firms with under 20 employees, and came from firms with under $1 million in annual revenues. The largest industry sectors were manufacturing, information, finance, and professional. The ratio of respondents for demographic data to agile methods data was quite high. Early pilot testing indicated respondents would be hesitant to provide demographic data. However, this was not the case after all.

Table 14. Demographic Data on Agile Methods, Benefits, and Website Quality

Demographic	Scale	Attribute	Response (%)
Job function	1	Executive	16 (6%)
	2	Director	15 (6%)
	3	Sr. Manager	27 (11%)
	4	Project Manager	37 (15%)
	5	Chief Engineer	11 (4%)
	6	Systems Engineer	13 (5%)
	7	Webmaster	5 (2%)
	8	Network Engineer	0 (0%)
	9	Database Admin	2 (1%)
	10	Program Analyst	13 (5%)
	11	Software Engineer	80 (32%)
	12	System Admin	5 (2%)
	13	Other	26 (10%)
	Total		**250 (100%)**
Years of experience	1	0-5	33 (13%)
	2	6-10	45 (18%)
	3	11-15	56 (22%)
	4	16-20	41 (16%)
	5	21-25	37 (15%)
	6	26-30	18 (7%)
	7	31-35	13 (5%)
	8	36-More	9 (4%)
	Total		**252 (100%)**
Number of employees	1	1-19	43 (18%)
	2	20-49	24 (10%)
	3	50-99	22 (9%)
	4	100-249	26 (11%)
	5	250-499	24 (10%)
	6	500-999	16 (7%)
	7	1,000-2,499	21 (9%)
	8	2,500-4,999	16 (7%)
	9	5,000-9,999	11 (4%)
	10	10,000-24,999	12 (5%)
	11	25,000-49,999	9 (4%)
	12	50,000-99,999	6 (2%)

Demographic	Scale	Attribute	Response (%)
	13	100,000-More	15 (6%)
	Total		**245 (100%)**
Annual revenue	1	0-$999,000	35 (22%)
	2	$1,000,000-$9,999,999	30 (19%)
	3	$10,000,000-$24,999,999	18 (11%)
	4	$25,000,000-$49,999,999	14 (9%)
	5	$50,000,000-$99,999,999	12 (7%)
	6	$100,000,000-$249,999,999	8 (5%)
	7	$250,000,000-$499,999,999	8 (5%)
	8	$500,000,000-$999,999,999	5 (3%)
	9	$1,000,000,000-More	31 (19%)
	Total		**161 (100%)**
Industry sector	1	Agriculture	1 (0%)
	2	Mining	1 (0%)
	3	Utilities	2 (1%)
	4	Construction	0 (0%)
	5	Manufacturing	28 (11%)
	6	Wholesale	2 (1%)
	7	Retail	7 (3%)
	8	Transportation	3 (1%)
	9	Information	52 (21%)
	10	Finance	22 (9%)
	11	Real Estate	2 (1%)
	12	Professional	63 (26%)
	13	Management	5 (2%)
	14	Administrative	1 (0%)
	15	Educational	15 (6%)
	16	Health Care	8 (3%)
	17	Arts	8 (3%)
	18	Accommodation	1 (0%)
	19	Other	18 (7%)
	20	Public Admin	7 (3%)
	Total		**230 (100%)**

Agile Methods Data

Summary analysis. A summary analysis of the data from the main survey instrument of agile methods was performed (as shown in Table 15). The data were grouped by the four major factors of agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. It was further broken down into the 20 variables associated with the four major factors of agile methods, five variables per factor. Finally, the data were organized horizontally according to a seven-point Likert-type scale used to solicit the responses: (a) strongly disagree, (b) disagree, (c) somewhat disagree, (d) neutral, (e) somewhat agree, (f) agree, and (g) strongly agree. The number of responses per variable was also shown. A score for the responses to each variable was shown as well. Two pieces of data were shown for each combination of variable and scale-item: (a) the number of responses per scale item and (b) the percentage of the total number of responses for each variable represented. For iterative development, the majority of the respondents answered with strongly agree, with the exception of operational releases. For customer feedback, the majority of respondents strongly agreed with feedback solicited, agreed with feedback received, somewhat agreed with feedback frequency, agreed with feedback quality, and strongly agreed with feedback incorporated. This revealed a minor trend in which feedback is solicited from customers, but not always received, not always received frequently, and is not always of the best quality. For well-structured teams, the majority of the respondents agreed with team leader, vision and strategy, goals and objectives, and schedules and timelines, while strongly agreeing with small team size. For flexibility, small size should have been better phrased as small change, and portable design should have been qualified with “Java virtual machine” in parentheses to strengthen these responses. These data generally show the ability of the survey instrument to measure compliance with agile methods.

Table 15. Agile Methods Data Summary

Factor	Variable	1 Strongly Disagree	2 Disagree	3 Somewhat Disagree	4 Neutral	5 Somewhat Agree	6 Agree	7 Strongly Agree	Responses	Score
ITERATIVE DEVELOPMENT	Time-boxed releases	14 (6%)	23 (9%)	22 (9%)	13 (5%)	38 (15%)	59 (24%)	**80 (32%)**	**249**	5.15 / 7 (73.55%)
	Operational releases	10 (4%)	18 (7%)	19 (8%)	17 (7%)	50 (20%)	**84 (34%)**	51 (20%)	**249**	5.15 / 7 (73.55%)
	Small releases	10 (4%)	24 (10%)	18 (7%)	25 (10%)	34 (14%)	64 (26%)	**71 (29%)**	**246**	5.13 / 7 (73.34%)
	Frequent releases	27 (11%)	30 (12%)	17 (7%)	23 (9%)	34 (14%)	50 (20%)	**66 (27%)**	**247**	4.70 / 7 (67.21%)
	Numerous releases	9 (4%)	18 (7%)	11 (4%)	26 (10%)	35 (14%)	67 (27%)	**82 (33%)**	**248**	5.38 / 7 (76.79%)
CUSTOMER FEEDBACK	Feedback solicited	6 (2%)	14 (6%)	12 (5%)	20 (8%)	38 (15%)	72 (29%)	**87 (35%)**	**249**	5.55 / 7 (79.23%)
	Feedback received	4 (2%)	16 (6%)	24 (10%)	17 (7%)	55 (22%)	**77 (31%)**	56 (22%)	**249**	5.24 / 7 (74.87%)
	Feedback frequency	7 (3%)	34 (14%)	35 (14%)	37 (15%)	**60 (24%)**	44 (18%)	29 (12%)	**246**	4.45 / 7 (63.59%)
	Feedback quality	25 (10%)	35 (14%)	39 (16%)	37 (15%)	44 (18%)	**45 (18%)**	21 (9%)	**246**	4.05 / 7 (57.90%)
	Feedback incorporated	6 (2%)	8 (3%)	12 (5%)	13 (5%)	44 (18%)	80 (32%)	**86 (35%)**	**249**	5.67 / 7 (81.01%)
WELL-STRUCTURED TEAMS	Team leader	9 (4%)	11 (4%)	20 (8%)	27 (11%)	43 (17%)	**84 (33%)**	57 (23%)	**251**	5.25 / 7 (74.96%)
	Vision and strategy	11 (4%)	20 (8%)	23 (9%)	29 (12%)	60 (24%)	**79 (32%)**	28 (11%)	**250**	4.82 / 7 (68.91%)
	Goals and objectives	3 (1%)	12 (5%)	22 (9%)	24 (10%)	57 (23%)	**95 (38%)**	36 (14%)	**249**	5.20 / 7 (74.35%)
	Schedules and timelines	6 (2%)	15 (6%)	26 (11%)	25 (10%)	59 (24%)	**67 (27%)**	48 (20%)	**246**	5.07 / 7 (72.42%)
	Small team size	6 (2%)	10 (4%)	10 (4%)	13 (5%)	16 (6%)	63 (25%)	**131 (53%)**	**249**	5.96 / 7 (85.08%)
FLEXIBILITY	Small size	13 (5%)	44 (18%)	18 (7%)	**61 (24%)**	51 (20%)	38 (15%)	24 (10%)	**249**	4.22 / 7 (60.24%)
	Simple design	8 (3%)	28 (11%)	21 (8%)	37 (15%)	47 (19%)	**67 (27%)**	41 (16%)	**249**	4.82 / 7 (68.79%)
	Modular design	6 (2%)	11 (4%)	14 (6%)	23 (9%)	49 (20%)	**87 (35%)**	58 (23%)	**248**	5.38 / 7 (76.90%)
	Portable design	38 (15%)	41 (17%)	20 (8%)	32 (13%)	36 (15%)	**43 (17%)**	38 (15%)	**248**	4.08 / 7 (58.29%)
	Extensible design	4 (2%)	16 (6%)	23 (9%)	23 (9%)	51 (20%)	**75 (30%)**	59 (24%)	**251**	5.24 / 7 (74.84%)

Correlational analysis. A correlational analysis of the 20 variables of agile methods was performed (as shown in Table 16). There were five variables associated with each of the four major factors of agile methods. As expected the five variables associated with each of the four major factors of agile methods were closely correlated (using Pearson correlations). The four major factors of agile methods are: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. Within the first group, iterative development, the highest correlation was between small releases and operational releases, small releases and frequent releases, and small releases and numerous releases. Within the second group, customer feedback, the highest correlations were between feedback solicited and feedback received, feedback solicited and feedback incorporated, and feedback frequency and feedback quality. Within the last two groups, well-structured teams and flexibility, the highest correlations were between vision and strategy and goals and objectives, and small size and simple design. With some exceptions, this analysis indicates the variables of agile methods were well-chosen.

Variable analysis. A linear regression of the 20 variables of agile methods was also performed (as shown in Table 17). Within the first group, iterative development, the highest adjusted R^2 values were between operational releases and small releases, small releases and frequent releases, and small releases and numerous releases. Within the second group, customer feedback, the highest adjusted R^2 values were between feedback solicited and feedback received, feedback received and feedback frequency, and feedback frequency and feedback quality. Within the third group, well-structured teams, the highest adjusted R^2 values were between vision and strategy and goals and objectives. Within the fourth group, flexibility, the highest adjusted R^2 values were between small size and simple design and modular design and extensible design. Overall, the highest adjusted R^2 values seem to be within the second group, customer feedback, but each major group of variables is closely related. This analysis further indicates the variables were well-chosen and reliably describe and represent the individual factors of agile methods.

Table 16. Agile Methods Data Correlational Analysis (Pearson Correlations)

Variable	Time-boxed releases	Operational releases	Small releases	Frequent releases	Numerous releases	Feedback solicited	Feedback received	Feedback frequency	Feedback quality	Feedback incorporated	Team leader	Vision and strategy	Goals and objectives	Schedules & timelines	Small team size	Small size	Simple design	Modular design	Portable design	Extensible design
Time-boxed releases	**1.000**	0.391	0.511	0.522	0.437	0.328	0.264	0.313	0.272	0.311	0.312	0.306	0.271	0.333	*-0.008*	0.238	0.237	0.353	0.201	0.273
Operational releases	0.391	**1.000**	0.558	0.373	0.446	0.237	0.296	0.204	0.344	0.337	0.242	0.267	0.256	*0.126*	0.157	0.277	0.286	0.277	*0.065*	0.238
Small releases	0.511	0.558	**1.000**	0.599	0.570	0.355	0.335	0.299	0.415	0.412	0.229	0.338	0.381	0.270	*0.144*	0.390	0.416	0.433	0.233	0.416
Frequent releases	0.522	0.373	0.599	**1.000**	0.458	0.255	0.265	0.322	0.358	0.213	0.222	0.369	0.371	0.309	*0.070*	0.320	0.330	0.430	0.295	0.413
Numerous releases	0.437	0.446	0.570	0.458	**1.000**	0.353	0.376	0.231	0.286	0.381	0.200	0.285	0.363	0.159	0.147	0.205	0.241	0.359	0.143	0.330
Feedback solicited	0.328	0.237	0.355	0.255	0.353	**1.000**	0.742	0.567	0.536	0.733	0.208	0.255	0.247	0.153	0.173	0.339	0.325	0.336	0.194	0.335
Feedback received	0.264	0.296	0.335	0.265	0.376	0.742	**1.000**	0.688	0.667	0.682	0.245	0.386	0.365	0.252	0.142	0.300	0.330	0.359	0.137	0.376
Feedback frequency	0.313	0.204	0.299	0.322	0.231	0.567	0.688	**1.000**	0.758	0.509	0.331	0.464	0.396	0.348	*0.018*	0.319	0.304	0.284	0.237	0.321
Feedback quality	0.272	0.344	0.415	0.358	0.286	0.536	0.667	0.758	**1.000**	0.494	0.265	0.421	0.399	0.253	*0.078*	0.335	0.383	0.318	0.257	0.370
Feedback incorporated	0.311	0.337	0.412	0.213	0.381	0.733	0.682	0.509	0.494	**1.000**	0.251	0.285	0.298	0.162	0.223	0.314	0.405	0.438	0.185	0.391
Team leader	0.312	0.242	0.229	0.222	0.200	0.208	0.245	0.331	0.265	0.251	**1.000**	0.600	0.540	0.524	*-0.011*	0.228	0.225	0.310	0.236	0.256
Vision and strategy	0.306	0.267	0.338	0.369	0.285	0.255	0.386	0.464	0.421	0.285	0.600	**1.000**	0.798	0.550	*0.078*	0.395	0.420	0.350	0.271	0.442
Goals and objectives	0.271	0.256	0.381	0.371	0.363	0.247	0.365	0.396	0.399	0.298	0.540	0.798	**1.000**	0.550	*0.087*	0.370	0.391	0.363	0.258	0.437
Schedules and timelines	0.333	*0.126*	0.270	0.309	0.159	0.153	0.252	0.348	0.253	0.162	0.524	0.550	0.550	**1.000**	*-0.009*	0.209	0.183	0.294	0.137	0.245
Small team size	*-0.008*	0.157	*0.144*	*0.070*	0.147	0.173	0.142	*0.018*	*0.078*	0.223	*-0.011*	*0.078*	*0.087*	*-0.009*	**1.000**	0.177	0.204	*0.024*	*-0.114*	*0.109*
Small size	0.238	0.277	0.390	0.320	0.205	0.339	0.300	0.319	0.335	0.314	0.228	0.395	0.370	0.209	0.177	**1.000**	0.712	0.453	0.268	0.478
Simple design	0.237	0.286	0.416	0.330	0.241	0.325	0.330	0.304	0.383	0.405	0.225	0.420	0.391	0.183	0.204	0.712	**1.000**	0.487	0.345	0.588
Modular design	0.353	0.277	0.433	0.430	0.359	0.336	0.359	0.284	0.318	0.438	0.310	0.350	0.363	0.294	*0.024*	0.453	0.487	**1.000**	0.377	0.594
Portable design	0.201	*0.065*	0.233	0.295	0.143	0.194	0.137	0.237	0.257	0.185	0.236	0.271	0.258	0.137	*-0.114*	0.268	0.345	0.377	**1.000**	0.353
Extensible design	0.273	0.238	0.416	0.413	0.330	0.335	0.376	0.321	0.370	0.391	0.256	0.442	0.437	0.245	*0.109*	0.478	0.588	0.594	0.353	**1.000**

Note. Data in red italics are not significant.

Table 17. Agile Methods Data Variable Analysis (Adjusted R^2 Values)

Variable	Time-boxed releases	Operational releases	Small releases	Frequent releases	Numerous releases	Feedback solicited	Feedback received	Feedback frequency	Feedback quality	Feedback incorporated	Team leader	Vision and strategy	Goals and objectives	Schedules & timelines	Small team size	Small size	Simple design	Modular design	Portable design	Extensible design
Time-boxed releases	**1.000**	0.149	0.258	0.269	0.187	0.104	0.066	0.095	0.070	0.093	0.094	0.090	0.069	0.107	*-0.004*	0.053	0.052	0.121	0.036	0.071
Operational releases	0.149	**1.000**	0.309	0.136	0.196	0.052	0.084	0.038	0.115	0.110	0.055	0.067	0.062	0.012	0.021	0.073	0.078	0.073	*0.000*	0.053
Small releases	0.258	0.309	**1.000**	0.356	0.322	0.122	0.109	0.085	0.168	0.166	0.049	0.111	0.141	0.069	0.017	0.148	0.169	0.184	0.050	0.170
Frequent releases	0.269	0.136	0.356	**1.000**	0.206	0.061	0.067	0.100	0.125	0.041	0.045	0.133	0.134	0.092	*0.001*	0.098	0.105	0.181	0.083	0.167
Numerous releases	0.187	0.196	0.322	0.206	**1.000**	0.121	0.138	0.049	0.078	0.142	0.036	0.078	0.128	0.021	0.017	0.038	0.058	0.125	0.016	0.105
Feedback solicited	0.104	0.052	0.122	0.061	0.121	**1.000**	0.549	0.319	0.284	0.535	0.040	0.061	0.057	0.019	0.026	0.112	0.102	0.109	0.034	0.109
Feedback received	0.066	0.084	0.109	0.067	0.138	0.549	**1.000**	0.471	0.442	0.463	0.056	0.145	0.130	0.060	0.016	0.086	0.105	0.125	0.015	0.138
Feedback frequency	0.095	0.038	0.085	0.100	0.049	0.319	0.471	**1.000**	0.573	0.256	0.106	0.212	0.153	0.117	*-0.004*	0.098	0.089	0.077	0.052	0.100
Feedback quality	0.070	0.115	0.168	0.125	0.078	0.284	0.442	0.573	**1.000**	0.241	0.066	0.174	0.155	0.060	*0.002*	0.108	0.143	0.097	0.062	0.133
Feedback incorporated	0.093	0.110	0.166	0.041	0.142	0.535	0.463	0.256	0.241	**1.000**	0.059	0.078	0.085	0.022	0.046	0.095	0.160	0.188	0.030	0.149
Team leader	0.094	0.055	0.049	0.045	0.036	0.040	0.056	0.106	0.066	0.059	**1.000**	0.357	0.289	0.271	*-0.004*	0.048	0.047	0.092	0.052	0.062
Vision and strategy	0.090	0.067	0.111	0.133	0.078	0.061	0.145	0.212	0.174	0.078	0.357	**1.000**	0.636	0.300	*0.002*	0.152	0.173	0.119	0.069	0.192
Goals and objectives	0.069	0.062	0.141	0.134	0.128	0.057	0.130	0.153	0.155	0.085	0.289	0.636	**1.000**	0.299	*0.033*	0.134	0.149	0.128	0.063	0.188
Schedules and timelines	0.107	0.012	0.069	0.092	0.021	0.019	0.060	0.117	0.060	0.022	0.271	0.300	0.299	**1.000**	*-0.004*	0.040	0.030	0.082	0.015	0.056
Small team size	*-0.004*	0.021	0.017	*0.001*	0.017	0.026	0.016	*-0.004*	*0.002*	0.046	*-0.004*	*0.002*	*0.033*	*-0.004*	**1.000**	0.027	0.038	*-0.004*	*0.009*	*0.008*
Small size	0.053	0.073	0.148	0.098	0.038	0.112	0.086	0.098	0.108	0.095	0.048	0.152	0.134	0.040	0.027	**1.000**	0.505	0.202	0.068	0.225
Simple design	0.052	0.078	0.169	0.105	0.058	0.102	0.105	0.089	0.143	0.160	0.047	0.173	0.149	0.03	0.038	0.505	**1.000**	0.234	0.115	0.343
Modular design	0.121	0.073	0.184	0.181	0.125	0.109	0.125	0.077	0.097	0.188	0.092	0.119	0.128	0.082	*-0.004*	0.202	0.234	**1.000**	0.138	0.350
Portable design	0.036	*0.000*	0.050	0.083	0.016	0.034	0.015	0.052	0.062	0.030	0.052	0.069	0.063	0.015	*0.009*	0.068	0.115	0.138	**1.000**	0.121
Extensible design	0.071	0.053	0.170	0.167	0.105	0.109	0.138	0.100	0.133	0.149	0.062	0.192	0.188	0.056	*0.008*	0.225	0.343	0.350	0.121	**1.000**

Note. Data in red italics are not significant.

Factor analysis. An analysis of the four major factors of agile methods was performed (as shown in Table 18). Linear regression was used to build statistical models between each factor. The four major factors of agile methods were found to be related to one another using this analysis: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. This analysis was designed to see whether the major factors of agile methods were related, whereas prior analysis was used to study the relationships between individual variables. The strongest adjusted R2 values were between iterative development and flexibility, customer feedback and flexibility, and well-structured teams and flexibility. The F-values were good and the significance was high, with no value exceeding zero. The lowest adjusted R2 values were between iterative development and customer feedback, iterative development and well-structured teams, and customer feedback and well-structured teams. This analysis indicates the individual factors are related to one another and reliably represent the major elements of agile methods.

Table 18. Agile Methods Factor Analysis

Factor	Statistic	Iterative development	Customer feedback	Well-structured teams	Flexibility
Iterative development	*Adjusted R^2 value*	**1.000**	0.231	0.185	0.284
	F-value	**n/a**	15.071	11.602	19.656
	Significance	**n/a**	0.000	0.000	0.000
Customer feedback	*Adjusted R^2 value*	0.220	**1.000**	0.201	0.230
	F-value	14.577	**n/a**	13.039	15.357
	Significance	0.000	**n/a**	0.000	0.000
Well-structured teams	*Adjusted R^2 value*	0.194	0.192	**1.000**	0.241
	F-value	12.599	12.422	**n/a**	16.286
	Significance	0.000	0.000	**n/a**	0.000
Flexibility	*Adjusted R^2 value*	0.258	0.227	0.233	**1.000**
	F-value	17.674	15.069	15.491	**n/a**
	Significance	0.000	0.000	0.000	**n/a**

Model analysis. Four statistical models were constructed between the four major factors of agile methods and the remaining factors (as shown in Table 19). All four factors of agile methods: iterative development, customer feedback, well-structured teams, and flexibility, had equally high adjusted R^2 values. Furthermore, the F-values and significance of the models were also high. Beta values for each of the independent variables were moderately high, the t-values were good, and the significance for each of the independent variables was high. The strongest model was flexibility as a function of iterative development, customer feedback, and well-structured teams. The weakest model was well-structured teams as a function of iterative development, customer feedback, and flexibility. This analysis indicates that the factor, variable, and item selection for agile methods was more than adequate for measuring compliance with agile methods. This analysis indicates the aggregated factors are related to one another and Table 16 through Table 19 indicate the main survey instrument of agile methods is reliable and valid.

Table 19. Agile Methods Model Analysis

Model	Statistic	Iterative development	Customer feedback	Well-structured teams	Flexibility
(*Model*)	*Adjusted R^2*	0.339	0.336	0.325	0.358
	F-value	43.062	42.571	40.490	46.787
	Significance	0.000	0.000	0.000	0.000
(*Constant*)	*Beta*	1.007	0.992	2.362	0.921
	t-value	2.623	2.633	8.751	2.692
	Significance	0.009	0.009	0.000	0.008
Iterative development	*Beta*	n/a	0.251	0.180	0.222
	t-value	n/a	4.112	3.618	3.995
	Significance	n/a	0.000	0.000	0.000
Customer feedback	*Beta*	0.260	n/a	0.171	0.233
	t-value	4.112	n/a	3.367	4.145
	Significance	0.000	n/a	0.001	0.000
Well-structured teams	*Beta*	0.284	0.261	n/a	0.296
	t-value	3.618	3.367	n/a	4.264
	Significance	0.000	0.001	n/a	0.000
Flexibility	*Beta*	0.278	0.283	0.235	n/a
	t-value	3.995	4.415	4.264	n/a
	Significance	0.000	0.000	0.000	n/a

Benefit Data

Summary analysis. A summary analysis of the benefit data was performed (as shown in Table 20). The estimates were rather conservative and generally spread evenly across all of the categories. In fact, the data tend to be skewed toward the lower end of the scale. (Note that the agile methods data is skewed toward the upper end of the scale.) The largest group of responses rated cost, quality, and cycle time benefits in the 0 to 10% range. Productivity benefits were rated in the 11% to 25% range. Customer satisfaction benefits were rated in the 75% to 100% range. The average score for the variables and scale-items was just under 50%. One observation is that 25% of the respondents cited a range of customer satisfaction benefits almost twice as high as the other four benefit variables. While the data are slightly skewed toward the lower end of the benefit scale, it is important to note that the average response is about 50%. What this means is that half of the respondents cited benefits for agile methods in excess of 50%. About half of the respondents reported data relating to the benefits of agile methods, but when benefit data were reported, half of these reported benefits in excess of 50% for each criterion.

Table 20. Benefit Data Summary

Variable	1 0 to 10%	2 11% to 25%	3 26% to 50%	4 51% to 75%	5 76% to 100%	6 101% to 200%	7 201% or more	Responses	Score
Cost	**31 (26%)**	23 (19%)	23 (19%)	18 (15%)	9 (7%)	14 (11%)	4 (3%)	**122**	3.07 / 7 (43.91%)
Productivity	27 (18%)	**28 (19%)**	24 (17%)	27 (19%)	19 (13%)	16 (11%)	4 (3%)	**145**	3.32 / 7 (47.49%)
Quality	**30 (20%)**	23 (15%)	28 (19%)	19 (13%)	19 (13%)	23 (15%)	7 (5%)	**149**	3.48 / 7 (49.66%)
Cycle Time	**26 (19%)**	24 (17%)	18 (13%)	25 (18%)	19 (14%)	15 (11%)	11 (8%)	**138**	3.55 / 7 (50.72%)
Customer Satisfaction	25 (18%)	14 (10%)	24 (18%)	14 (10%)	**33 (25%)**	15 (11%)	11 (8%)	**136**	3.77 / 7 (53.89%)

Correlational analysis. A correlational analysis of the five benefit variables was performed (as shown in Table 21). The five benefit variables were: (a) cost, (b) productivity, (c) quality, (d) cycle time, and (e) customer satisfaction. The five benefit variables were closely correlated (using Pearson correlations). In fact, all of the Pearson correlations were extremely high, with the lowest Pearson correlation being 0.748. The strongest correlation was between the benefit variables of cost and productivity. The weakest correlation was between the benefit variables of quality and cycle time. Most of the correlations were extremely close in range and there were no consistently high or low patterns of correlations between variables. Sometimes, benefit variables are considered to be orthogonal in nature (e.g., cost and quality). For instance, quality may be considered expensive to obtain. Therefore, one benefit variable has to be traded off for another. Surprisingly, these correlations revealed no such perceptions of benefit variables. These data indicate unusually high levels of agreement between respondents about the benefits of agile methods and may be worthy of further investigation in future research studies.

Table 21. Benefit Data Correlational Analysis (Pearson Correlations)

Variable	Statistic	Cost	Productivity	Quality	Cycle Time	Customer Satisfaction
Cost	*Pearson*	**1.000**	0.902	0.824	0.839	0.793
	Significance	**n/a**	0.000	0.000	0.000	0.000
	N	**122**	121	120	113	113
Productivity	*Pearson*	0.902	**1.000**	0.769	0.829	0.785
	Significance	0.000	**n/a**	0.000	0.000	0.000
	N	121	**145**	144	133	130
Quality	*Pearson*	0.824	0.769	**1.000**	0.748	0.854
	Significance	0.000	0.000	**n/a**	0.000	0.000
	N	120	144	**149**	136	134
Cycle Time	*Pearson*	0.839	0.829	0.748	**1.000**	0.795
	Significance	0.000	0.000	0.000	**n/a**	0.000
	N	113	133	136	**138**	129
Customer Satisfaction	*Pearson*	0.793	0.785	0.854	0.795	**1.000**
	Significance	0.000	0.000	0.000	0.000	**n/a**
	N	113	130	134	129	**136**

Variable analysis. A linear regression of the five benefit variables was also performed (as shown in Table 22). This analysis exhibited the highest adjusted R^2 values, F-values, and significance values found in this study. For instance, the highest adjusted R^2 value was 0.812, between the benefit variables of cost and productivity. This means that more than 81% of the variation is explained by these two variables. The lowest adjusted R^2 value was 0.557, between the benefit variables of quality and cycle time. While 56% of the variation is explained between the variables of quality and cycle time, which is very respectable, this is a stark contrast to the earlier value of 81%. This may indicate perceived tradeoffs between major classes of benefit variables. For instance, there may be some lingering perceptions that cycle time must be sacrificed for quality. Even if this were so, a 56% adjusted R^2 value is still very high, which may indicate that perceptions about tradeoffs involving the high cost-of-quality may be dissipating. This analysis indicates that the respondents unanimously agreed with statements concerning the benefits of agile methods and the data about the benefits of agile methods are reliable and valid.

Table 22. Benefit Data Variable Analysis (Adjusted R^2 Values)

Variable	Statistic	Cost	Productivity	Quality	Cycle Time	Customer Satisfaction
Cost	*Adjusted R^2*	**1.000**	0.812	0.676	0.701	0.625
	F-value	**n/a**	519.787	248.762	263.247	187.784
	Significance	**n/a**	0.000	0.000	0.000	0.000
Productivity	*Adjusted R^2*	0.812	**1.000**	0.589	0.685	0.614
	F-value	519.787	**n/a**	205.707	274.472	205.977
	Significance	0.000	**n/a**	0.000	0.000	0.000
Quality	*Adjusted R^2*	0.676	0.589	**1.000**	0.557	0.727
	F-value	248.762	205.707	**n/a**	170.435	354.390
	Significance	0.000	0.000	**n/a**	0.000	0.000
Cycle Time	*Adjusted R^2*	0.701	0.685	0.557	**1.000**	0.629
	F-value	263.247	274.472	170.435	**n/a**	218.351
	Significance	0.000	0.000	0.000	**n/a**	0.000
Customer Satisfaction	*Adjusted R^2*	0.625	0.614	0.727	0.629	**1.000**
	F-value	187.784	205.977	354.390	218.351	**n/a**
	Significance	0.000	0.000	0.000	0.000	**n/a**

Model analysis. Five statistical models were constructed between the five benefit variables and the remaining variables (as shown in Table 23). All five benefit variables—cost, productivity, quality, cycle time, and customer satisfaction—had high adjusted R^2 values. Furthermore, the F-values and significance of the models was extremely high. Four out of five statistical models had high adjusted R^2 values, which explained 85% of the variation between the variables. Only one adjusted R^2 value was substantially below the 85% threshold. Cycle time had an adjusted R^2 value of almost 80%, which is very high. The F-values for these were some of the highest in this study. There were some minor breakdowns in the significance of the individual Beta values (shown in italics). This analysis was performed to test the reliability and validity of the data for later analysis of the potential relationships between agile methods and their benefits. The high adjusted R^2 values in the top row provide strong evidence that agile methods are related to improvements in cost efficiency, productivity, quality, cycle time, and customer satisfaction.

Table 23. Benefit Data Model Analysis

Variable	Statistic	Cost	Productivity	Quality	Cycle Time	Customer Satisfaction
(*Model*)	*Adjusted R^2*	0.854	0.863	0.848	0.786	0.852
	F-value	157.912	169.476	149.692	99.504	155.410
	Significance	0.000	0.000	0.000	0.000	0.000
(*Constant*)	*Beta*	-0.136	0.201	0.111	0.166	0.276
	t-value	-0.897	1.392	0.683	0.839	1.720
	Significance	*0.372*	*0.167*	*0.496*	*0.403*	*0.088*
Cost	*Beta*	**1.000**	0.573	0.194	0.371	-0.066
	t-value	**n/a**	7.578	1.872	3.021	-0.623
	Significance	**n/a**	0.000	*0.064*	0.003	*0.534*
Productivity	*Beta*	0.625	**1.000**	0.225	0.322	0.026
	t-value	7.578	**n/a**	2.088	2.470	0.233
	Significance	0.000	**n/a**	0.039	0.015	*0.816*
Quality	*Beta*	0.170	0.180	**1.000**	-0.197	0.686
	t-value	1.872	2.088	**n/a**	-1.662	9.558
	Significance	*0.064*	0.039	**n/a**	*0.100*	0.000
Cycle Time	*Beta*	0.219	0.174	-0.133	**1.000**	0.331
	t-value	3.021	2.470	-1.662	**n/a**	4.473
	Significance	0.003	0.015	*0.100*	**n/a**	0.000
Customer Satisfaction	*Beta*	-0.057	0.021	0.685	0.491	**1.000**
	t-value	-0.623	0.233	9.558	4.473	**n/a**
	Significance	*0.534*	*0.816*	0.000	0.000	**n/a**

Note. Data in red italics are not significant.

Website Quality Data

Summary analysis. A summary analysis of the data from the measurement instrument for website quality was performed (as shown in Table 24). The data were grouped by the four major factors of website quality: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. It was further broken down into the 14 variables associated with the four major factors of website quality. Finally, the data were organized horizontally according to a seven-point Likert-type scale used to solicit the responses: (a) strongly disagree, (b) disagree, (c) somewhat disagree, (d) neutral, (e) somewhat agree, (f) agree, and (g) strongly agree. The number of responses per variable was also shown. A score for the responses to each variable was shown as well. Two pieces of data were shown for each combination of variable and scale-item: (a) the number of responses per scale item and (b) the percentage of the total number of responses for each variable represented. For website design, the majority of the respondents answered with agree, and strongly agree for processing speed. For privacy and security, the majority of respondents strongly agreed with protection and privacy, feelings of safety, and adequate security. For fulfillment and reliability, the majority of the responses were equally assigned to agree and strongly agree for order received, on-time delivery, and order accurate. For customer service, the majority of respondents answered strongly agree for willingness to respond, desire to fix issues, and promptness of service. These data generally show the ability of the measurement instrument to gauge perceptions of website quality. The data were arranged from left to right to show the degree to which the evaluator strongly disagreed, disagreed, somewhat disagreed, was neutral, somewhat agreed, agreed, or strongly agreed with perceptions of website quality. As shown, the largest issues seem to be slow processing speed, inadequate product selection, and in some cases inadequate security and failure to deliver.

Table 24. Website Quality Data Summary

Factor	Variable	1 Strongly Disagree	2 Disagree	3 Somewhat Disagree	4 Neutral	5 Somewhat Agree	6 Agree	7 Strongly Agree	Responses	Score
WEBSITE DESIGN	In-depth information	0 (0%)	0 (0%)	0 (0%)	0 (0%)	1 (10%)	**7 (70%)**	2 (20%)	**10**	6.1 / 7 (87.14%)
	Processing efficiency	0 (0%)	0 (0%)	1 (10%)	0 (0%)	2 (20%)	**4 (40%)**	3 (30%)	**10**	5.8 / 7 (82.86%)
	Processing speed	0 (0%)	2 (20%)	1 (10%)	1 (10%)	0 (0%)	**3 (30%)**	**3 (30%)**	**10**	5.0 / 7 (71.43%)
	Personalization	0 (0%)	0 (0%)	1 (10%)	2 (20%)	0 (0%)	**4 (40%)**	3 (30%)	**10**	5.6 / 7 (80.00%)
	Product selection	0 (0%)	1 (10%)	0 (0%)	0 (0%)	0 (0%)	**7 (70%)**	2 (20%)	**10**	5.8 / 7 (82.86%)
PRIVACY AND SECURITY	Protection of privacy	0 (0%)	0 (0%)	0 (0%)	3 (30%)	0 (0%)	1 (10%)	**6 (60%)**	**10**	6.0 / 7 (85.71%)
	Feelings of safety	0 (0%)	0 (0%)	0 (0%)	3 (30%)	0 (0%)	1 (10%)	**6 (60%)**	**10**	6.0 / 7 (85.71%)
	Adequate security	0 (0%)	1 (10%)	0 (0%)	3 (30%)	0 (0%)	1 (10%)	**5 (50%)**	**10**	5.5 / 7 (78.57%)
FULFILLMENT AND RELIABILITY	Order received	1 (10%)	0 (0%)	1 (10%)	1 (10%)	1 (10%)	**3 (30%)**	**3 (30%)**	**10**	5.2 / 7 (74.29%)
	On time delivery	1 (10%)	0 (0%)	1 (10%)	1 (10%)	1 (10%)	**3 (30%)**	**3 (30%)**	**10**	5.2 / 7 (74.29%)
	Order accurate	1 (10%)	0 (0%)	1 (10%)	1 (10%)	1 (10%)	**3 (30%)**	**3 (30%)**	**10**	5.2 / 7 (74.29%)
CUSTOMER SERVICE	Willingness to respond	1 (10%)	0 (0%)	0 (0%)	1 (10%)	1 (10%)	2 (20%)	**5 (50%)**	**10**	5.7 / 7 (81.43%)
	Desire to fix issues	1 (10%)	0 (0%)	0 (0%)	1 (10%)	1 (10%)	2 (20%)	**5 (50%)**	**10**	5.7 / 7 (81.43%)
	Promptness of service	1 (10%)	0 (0%)	0 (0%)	1 (10%)	1 (10%)	2 (20%)	**5 (50%)**	**10**	5.7 / 7 (81.43%)

Correlational analysis. A correlational analysis of the 14 variables of website quality was performed (as shown in Table 25). There were five variables associated with the first factor and three variables associated the last three factors of website quality. Many of the variables associated with each of the four major factors of website quality were closely correlated (using Pearson correlations). The four major factors of website quality are: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. Within the first group, website design, the highest correlations were between in-depth information and processing efficiency, processing efficiency and processing speed, and processing efficiency and personalization. Within the second group, privacy and security, the highest correlation was between protection of privacy and feelings of safety. Within the last two groups, fulfillment and reliability and customer service, all of the variables had a surprisingly high one-to-one correlation. With a few notable exceptions, this analysis indicates the variables of website quality were closely related within categories of variables, but not between categories of variables.

Variable analysis. A linear regression of the 14 variables of website quality was also performed (as shown in Table 26). Within the first group, website design, the highest adjusted R^2 values were between in-depth information and processing efficiency, processing efficiency and processing speed, and processing efficiency and personalization. Within the second group, privacy and security, the highest adjusted R^2 values were between protection of privacy and feelings of safety. Within the third group, fulfillment and reliability, all of the variables had a surprisingly high one-to-one correlation. Within the fourth group, customer service, all of the variables also had a surprisingly high one-to-one correlation. Overall, most, if not all of the groups had high adjusted R^2 values. This was especially true for the last two groups. There seemed to be some problems with low correlations in the first and second groups, website design and privacy and security. Again, this analysis indicates the variables are correlated within individual categories and the website quality instrument exhibits basic reliability and validity.

Table 25. Website Quality Data Correlational Analysis (Pearson Correlations)

Variable	In-depth information	Processing efficiency	Processing speed	Personali-zation	Product selection	Protection of privacy	Feelings of safety	Adequate security	Order received	On time delivery	Order accurate	Willingness to respond	Desire to fix issues	Promptness of service
In-depth information	**1.000**	0.828	0.667	*0.602*	*0.308*	*0.554*	*0.554*	*0.478*	*0.571*	*0.571*	*0.571*	0.734	0.734	0.734
Processing efficiency	0.828	**1.000**	0.792	0.834	*0.362*	0.831	0.831	*0.589*	0.791	0.791	0.791	0.854	0.854	0.854
Processing speed	0.667	0.792	**1.000**	0.719	*0.309*	0.650	0.650	0.940	0.707	0.707	0.707	0.861	0.861	0.861
Personalization	*0.602*	0.834	0.719	**1.000**	0.734	0.934	0.934	0.633	*0.539*	*0.539*	*0.539*	*0.551*	*0.551*	*0.551*
Product selection	*0.308*	*0.362*	*0.309*	0.734	**1.000**	*0.562*	*0.562*	*0.388*	*-0.064*	*-0.064*	*-0.064*	*0.057*	*0.057*	*0.057*
Protection of privacy	*0.554*	0.831	0.650	0.934	*0.562*	**1.000**	1.000	*0.555*	*0.593*	*0.593*	*0.593*	*0.605*	*0.605*	*0.605*
Feelings of safety	*0.554*	0.831	0.650	0.934	*0.562*	1.000	**1.000**	*0.555*	*0.593*	*0.593*	*0.593*	*0.605*	*0.605*	*0.605*
Adequate security	*0.478*	*0.589*	0.940	0.633	*0.388*	*0.555*	*0.555*	**1.000**	*0.486*	*0.486*	*0.486*	0.729	0.729	0.729
Order received	*0.571*	0.791	0.707	*0.539*	*-0.064*	*0.593*	*0.593*	*0.486*	**1.000**	1.000	1.000	0.792	0.792	0.792
On time delivery	*0.571*	0.791	0.707	*0.539*	*-0.064*	*0.593*	*0.593*	*0.486*	1.000	**1.000**	1.000	0.792	0.792	0.792
Order accurate	*0.571*	0.791	0.707	*0.539*	*-0.064*	*0.593*	*0.593*	*0.486*	1.000	1.000	**1.000**	0.792	0.792	0.792
Willingness to respond	0.734	0.854	0.861	*0.551*	*0.057*	*0.605*	*0.605*	0.729	0.792	0.792	0.792	**1.000**	1.000	1.000
Desire to fix issues	0.734	0.854	0.861	*0.551*	*0.057*	*0.605*	*0.605*	0.729	0.792	0.792	0.792	1.000	**1.000**	1.000
Promptness of service	0.734	0.854	0.861	*0.551*	*0.057*	*0.605*	*0.605*	0.729	0.792	0.792	0.792	1.000	1.000	**1.000**

Note. Data in red italics are not significant.

Table 26. Website Quality Data Variable Analysis (Adjusted R^2 Values)

Variable	In-depth information	Processing efficiency	Processing speed	Personali-zation	Product selection	Protection of privacy	Feelings of safety	Adequate security	Order received	On time delivery	Order accurate	Willingness to respond	Desire to fix issues	Promptness of service
In-depth information	**1.000**	0.646	0.375	*0.283*	*-0.018*	*0.220*	*0.220*	*0.133*	*0.242*	*0.242*	*0.242*	0.481	0.481	0.481
Processing efficiency	0.646	**1.000**	0.580	0.658	*0.022*	0.652	0.652	*0.266*	0.578	0.578	0.578	0.696	0.696	0.696
Processing speed	0.375	0.580	**1.000**	0.456	*-0.017*	0.350	0.350	0.869	0.437	0.437	0.437	0.709	0.709	0.709
Personalization	*0.283*	0.658	0.456	**1.000**	0.538	0.857	0.857	0.326	*0.202*	*0.202*	*0.202*	*0.216*	*0.216*	*0.216*
Product selection	*-0.018*	*0.022*	*-0.017*	0.538	**1.000**	*0.230*	*0.230*	*0.045*	*-0.120*	*-0.120*	*-0.120*	*-0.121*	*-0.121*	*-0.121*
Protection of privacy	*0.220*	0.652	0.350	0.857	*0.230*	**1.000**	1.000	*0.221*	*0.270*	*0.270*	*0.270*	*0.287*	*0.287*	*0.287*
Feelings of safety	*0.220*	0.652	0.350	0.857	*0.230*	1.000	**1.000**	*0.221*	*0.270*	*0.270*	*0.270*	*0.287*	*0.287*	*0.287*
Adequate security	*0.133*	*0.266*	0.869	0.326	*0.045*	*0.221*	*0.221*	**1.000**	*0.140*	*0.140*	*0.140*	0.472	0.472	0.472
Order received	*0.242*	0.578	0.437	*0.202*	*-0.120*	*0.270*	*0.270*	*0.140*	**1.000**	1.000	1.000	0.581	0.581	0.581
On time delivery	*0.242*	0.578	0.437	*0.202*	*-0.120*	*0.270*	*0.270*	*0.140*	1.000	**1.000**	1.000	0.581	0.581	0.581
Order accurate	*0.242*	0.578	0.437	*0.202*	*-0.120*	*0.270*	*0.270*	*0.140*	1.000	1.000	**1.000**	0.581	0.581	0.581
Willingness to respond	0.481	0.696	0.709	*0.216*	*-0.121*	*0.287*	*0.287*	0.472	0.581	0.581	0.581	**1.000**	1.000	1.000
Desire to fix issues	0.481	0.696	0.709	*0.216*	*-0.121*	*0.287*	*0.287*	0.472	0.581	0.581	0.581	1.000	**1.000**	1.000
Promptness of service	0.481	0.696	0.709	*0.216*	*-0.121*	*0.287*	*0.287*	0.472	0.581	0.581	0.581	1.000	1.000	**1.000**

Note. Data in red italics are not significant.

Factor analysis. An analysis of the four major factors of website quality was performed (as shown in Table 27). Linear regression was used to build statistical models between each factor. The four major factors of website quality proved to be related to one another using this analysis: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. This analysis was designed to see whether the major factors of website quality were related, whereas prior analysis was used to study the relationships between individual variables. The strongest adjusted R^2 values were between customer service and website design, privacy and security and website design, and customer service and fulfillment and reliability. The F-values were good and the significance was also good in all but four instances. The lowest adjusted R^2 values were between fulfillment and reliability and privacy and security and website design and fulfillment and reliability. Two more factors, and perhaps three, are correlated at the 0.10 level, though the minimum threshold of reliability used in this study is the 0.05 level. This analysis indicates the website quality instrument is reliable and valid at the factor-level.

Table 27. Website Quality Factor Analysis

Factor	Statistic	Website Design	Privacy and Security	Fulfillment and Reliability	Customer Service
Website Design	*Adjusted R^2 value*	**1.000**	0.860	0.585	0.899
	F-value	**n/a**	12.086	3.535	17.058
	Significance	**n/a**	0.016	*0.122*	0.008
Privacy and Security	*Adjusted R^2 value*	0.831	**1.000**	0.211	0.472
	F-value	23.147	**n/a**	2.206	5.025
	Significance	0.001	**n/a**	*0.181*	0.044
Fulfillment and Reliability	*Adjusted R^2 value*	0.297	0.308	**1.000**	0.581
	F-value	4.808	5.002	**n/a**	13.477
	Significance	*0.060*	*0.056*	**n/a**	0.006
Customer Service	*Adjusted R^2 value*	0.480	0.486	0.581	**1.000**
	F-value	9.294	9.516	13.477	**n/a**
	Significance	0.016	0.015	0.006	**n/a**

Note. Data in red italics are not significant.

Model analysis. Four statistical models were constructed between the four major factors of website quality and the remaining factors (as shown in Table 28). Three out of four factors of website quality: website design, privacy and security, and customer service, had high adjusted R^2 values. Furthermore, the F-values and significance of these three models was also good. While Beta values for each of the independent variables seemed high, the t-values were low, and the significance for many of the independent variables was also low. The strongest model was privacy and security as a function of website design, fulfillment and reliability, and customer service. The weakest model was fulfillment and reliability as a function of website design, privacy and security, and customer service. This analysis indicates the factor, variable, and item selection for website quality were adequate. All models are valid at the 0.10 level of significance and Beta values become significant for three more factors. This analysis further indicates the aggregated factors are related to one another and Table 25 through Table 28 indicate the survey instrument for website quality is reliable and valid.

Table 28. Website Quality Model Analysis

Model	Statistic	Website Design	Privacy and Security	Fulfillment and Reliability	Customer Service
(*Model*)	*Adjusted R^2*	0.809	0.812	0.446	0.597
	F-value	13.709	13.941	3.411	5.447
	Significance	0.004	0.004	*0.094*	0.038
(*Constant*)	*Beta*	1.1275	-0.548	0.245	-0.814
	t-value	1.799	-0.521	0.092	-0.371
	Significance	*0.122*	*0.621*	*0.930*	*0.723*
Website Design	*Beta*	n/a	1.041	0.011	0.336
	t-value	n/a	3.959	0.009	0.359
	Significance	n/a	0.007	*0.993*	*0.732*
Privacy and Security	*Beta*	0.695	n/a	0.107	0.297
	t-value	3.959	n/a	0.106	0.356
	Significance	0.009	n/a	*0.919*	*0.734*
Fulfillment and Reliability	*Beta*	0.001	0.017	n/a	0.521
	t-value	0.001	0.106	n/a	1.959
	Significance	*0.993*	*0.919*	n/a	*0.098*
Customer Service	*Beta*	0.057	0.070	0.749	n/a
	t-value	0.359	0.356	1.959	n/a
	Significance	*0.732*	*0.734*	*0.098*	n/a

Note. Data in red italics are not significant.

Agile Methods and Benefits

Correlational analysis. A correlational analysis between the agile methods and benefit variables was performed (as shown in Table 29). There were 20 variables associated with the four major factors of agile methods and five benefit variables. Most of the agile methods and benefit variables were closely related using Pearson correlations. All of the variables associated with the agile factors of iterative development and customer feedback were correlated to all five benefit variables. Most of the variables associated with the agile factors of well-structured teams and flexibility were also correlated to all five benefit variables. High response ratings for small team size caused issues with its correlations, along with low response ratings for portable design. With a few exceptions, this analysis shows all agile methods and benefit variables were related (i.e., higher compliance with agile methods results in greater benefits across the board).

Table 29. Agile-Benefit Data Correlational Analysis (Pearson Correlations)

Factor	Variable	Cost	Productivity	Quality	Cycle time	Customer Satisfaction
Iterative development	Time-boxed releases	0.398	0.311	0.417	0.323	0.376
	Operational releases	0.403	0.409	0.454	0.388	0.387
	Small releases	0.409	0.413	0.439	0.425	0.378
	Frequent releases	0.350	0.254	0.419	0.326	0.310
	Numerous releases	0.369	0.279	0.369	0.243	0.308
Customer feedback	Feedback solicited	0.385	0.273	0.331	0.219	0.330
	Feedback received	0.472	0.336	0.372	0.301	0.338
	Feedback frequency	0.503	0.326	0.402	0.344	0.351
	Feedback quality	0.592	0.409	0.464	0.449	0.438
	Feedback incorporated	0.432	0.364	0.352	0.285	0.315
Well-structured teams	Team leader	0.239	0.232	0.225	0.204	0.173
	Vision and strategy	0.383	0.339	0.376	0.252	0.306
	Goals and objectives	0.380	0.322	0.417	0.278	0.367
	Schedules and timelines	0.232	*0.144*	0.233	0.174	0.175
	Small team size	*0.146*	0.187	0.194	*0.133*	*0.104*
Flexibility	Small size	0.401	0.370	0.412	0.223	0.280
	Simple design	0.415	0.406	0.487	0.321	0.383
	Modular design	0.375	0.331	0.411	0.305	0.303
	Portable design	0.185	*0.122*	0.238	*0.072*	*0.170*
	Extensible design	0.356	0.347	0.408	0.285	0.356

Note. Data in red italics are not significant.

Variable analysis. A linear regression of the agile methods and benefit variables was also performed (as shown in Table 30). Most of the agile methods and benefit variables had high adjusted R^2 values. The adjusted R^2 values between the iterative development, customer feedback, and benefit variables were the highest. In general, the adjusted R^2 values between the agile methods and cost and quality benefit variables were the highest. Once again, high response ratings for small team size caused problems with its adjusted R^2 values, along with low response ratings for portable design. The response ratings for small team size were so high, that it may warrant removal from the survey instrument and perhaps even this dataset to enhance the analysis. The opposite was true of portable design, because it had a low response rating. The intention of portable design was to represent the use of cross-platform Java applications. This analysis indicates the correlations between agile methods and their benefits were rather weak, averaging around 12%, whereas strong correlations would have been 30%, 60%, or even 90%.

Table 30. Agile-Benefit Data Variable Analysis (Adjusted R^2 Values)

Factor	Variable	Cost	Productivity	Quality	Cycle time	Customer Satisfaction
Iterative development	Time-boxed releases	0.151	0.090	0.168	0.097	0.135
	Operational releases	0.155	0.162	0.201	0.144	0.144
	Small releases	0.161	0.165	0.188	0.174	0.136
	Frequent releases	0.115	0.058	0.170	0.100	0.089
	Numerous releases	0.128	0.071	0.130	0.052	0.088
Customer feedback	Feedback solicited	0.141	0.068	0.103	0.041	0.102
	Feedback received	0.217	0.106	0.133	0.084	0.107
	Feedback frequency	0.247	0.100	0.156	0.112	0.117
	Feedback quality	0.345	0.161	0.210	0.196	0.186
	Feedback incorporated	0.180	0.127	0.118	0.074	0.093
Well-structured teams	Team leader	0.049	0.047	0.044	0.035	0.023
	Vision and strategy	0.140	0.109	0.135	0.057	0.087
	Goals and objectives	0.138	0.097	0.169	0.070	0.128
	Schedules and timelines	0.046	*0.014*	0.048	0.023	0.023
	Small team size	*0.013*	0.028	0.031	*0.010*	*0.003*
Flexibility	Small size	0.154	0.131	0.164	0.043	0.072
	Simple design	0.165	0.159	0.232	0.096	0.140
	Modular design	0.133	0.103	0.163	0.086	0.085
	Portable design	0.026	*0.008*	0.050	*-0.002*	*0.021*
	Extensible design	0.120	0.114	0.160	0.074	0.120

Note. Data in red italics are not significant.

Factor analysis. An analysis of the four major factors of agile methods and the five benefit variables was performed (as shown in Table 31). Linear regression was used to build statistical models between each factor of agile methods and benefit variables. The four major factors of agile methods proved to be related to the benefit variables using this analysis: (a) cost, (b) productivity, (c) quality, (d) cycle time, and (e) customer satisfaction. This analysis was designed to see whether the five benefit variables were related to the four factors of agile methods, whereas prior analysis was used to study the relationships between individual variables. The strongest adjusted R^2 values were between the benefit variables of cost and quality and the four factors of agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. The F-values were good and the significance was also good in all five instances. The adjusted R^2 values were interesting, but not extremely high (e.g., 90%). This analysis indicates the factors of agile methods only had an average correlation of 17% to the benefit variables, whereas the Pearson correlations in Table 29 indicated stronger correlations.

Table 31. Agile-Benefit Factor Analysis

Factor	Statistic	Cost	Productivity	Quality	Cycle Time	Customer Satisfaction
Iterative development	*Adjusted R^2 value*	0.211	0.184	0.290	0.196	0.187
	F-value	6.988	7.147	12.455	7.347	6.890
	Significance	0.000	0.000	0.000	0.000	0.000
Customer feedback	*Adjusted R^2 value*	0.346	0.161	0.193	0.170	0.157
	F-value	13.146	6.301	7.800	6.451	5.803
	Significance	0.000	0.000	0.000	0.000	0.000
Well-structured teams	*Adjusted R^2 value*	0.116	0.100	0.154	0.052	0.101
	F-value	4.074	4.097	6.224	2.465	3.938
	Significance	0.002	0.002	0.000	0.036	0.002
Flexibility	*Adjusted R^2 value*	0.171	0.169	0.241	0.098	0.137
	F-value	5.698	6.576	10.033	3.840	5.086
	Significance	0.000	0.000	0.000	0.003	0.000

Model analysis. Five statistical models were constructed between the five benefit variables and the four major factors of agile methods (as shown in Table 32). All five benefit variables—cost, productivity, quality, cycle time, and customer satisfaction—had high adjusted R^2 values. Furthermore, the F-values and significance of these five models was also good. There were mixed results for the Beta values of each of the independent variables. The Beta values for iterative development and customer feedback were high and significant. However, few of the Beta values for well-structured teams and flexibility were strongly correlated. The t-values were low and the significance for well-structured teams and flexibility was low. The strongest model was quality as a function of iterative development, customer feedback, well-structured teams, and flexibility. The weakest model was cycle time as a function of iterative development, customer feedback, well-structured teams, and flexibility. This analysis indicates the aggregated factors of agile methods have an average correlation to the benefit variables of 29%, which is ample evidence the variables are related, but not strongly (e.g., 60%, 90%, or greater).

Table 32. Agile-Benefit Model Analysis

Model	Statistic	Cost	Productivity	Quality	Cycle Time	Customer Satisfaction
(*Model*)	*Adjusted R^2*	0.361	0.242	0.377	0.238	0.250
	F-value	17.970	12.439	23.260	11.649	12.185
	Significance	0.000	0.000	0.000	0.000	0.000
(*Constant*)	*Beta*	-1.572	-0.807	-1.955	-1.103	-0.648
	t-value	-2.424	-1.244	-3.067	-1.386	-0.864
	Significance	0.017	*0.216*	0.003	*0.168*	*0.389*
Iterative development	*Beta*	0.248	0.277	0.433	0.467	0.372
	t-value	2.078	2.352	3.805	3.541	2.812
	Significance	0.040	0.020	0.000	0.001	0.006
Customer feedback	*Beta*	0.462	0.231	0.205	0.276	0.302
	t-value	4.052	2.042	1.853	2.164	2.340
	Significance	0.000	0.043	*0.066*	0.032	0.021
Well-structured teams	*Beta*	0.021	0.075	0.043	0.069	0.015
	t-value	0.150	0.549	0.315	0.438	0.097
	Significance	*0.881*	*0.584*	*0.754*	*0.662*	*0.923*
Flexibility	*Beta*	0.186	0.227	0.383	0.074	0.168
	t-value	1.493	1.881	3.188	0.553	1.241
	Significance	*0.138*	*0.062*	0.002	*0.581*	*0.217*

Note. Data in red italics are not significant.

Agile Methods and Website Quality

Correlational analysis. A correlational analysis between the variables of agile methods and website quality was performed (as shown in Table 33). There were 20 variables associated with the four major factors of agile methods: (a) iterative development, (b) customer feedback, (c) well-structured teams, and (d) flexibility. There were 14 variables associated with the four major factors of website quality: (a) website design, (b) privacy and security, (c) fulfillment and reliability, and (d) customer service. Few if any of the variables of agile methods and website quality were closely related using Pearson correlations. There were some spurious correlations between time-boxed releases and website design and privacy and security. There were more spurious correlations between small size and website quality. However, the majority of agile methods and website quality variables were not related using Pearson correlations. This is important, because the basic objective of this study was to help prove these relations existed. This analysis indicates the variables of agile methods are not related to website quality, even using Pearson correlations, which tend to be sensitive to minute correlations between variables.

Variable analysis. A linear regression between the variables of agile methods and website quality was also performed (as shown in Table 34). Few if any of the variables of agile methods and website quality had high adjusted R^2 values. There were some spuriously high adjusted R^2 values between time-boxed releases and website design and privacy and security. There were more spuriously high adjusted R^2 values between small size and website quality, particularly fulfillment and reliability. However, the majority of agile methods and website quality variables did not have high adjusted R^2 values in common. Once again, this is important, because the basic objective of this study was to help prove these relations existed. These weak relationships are primarily due to the small volume of data collected for website quality. Had self-reported data on website quality been collected, as it had for benefit data, the relationships between the variables may have been stronger. This analysis indicates the variables of agile methods and website quality had little or no strong correlations.

Table 33. Agile-Website Quality Data Correlational Analysis (Pearson Correlations)

Variable	In-depth information	Processing efficiency	Processing speed	Personali-zation	Product selection	Protection of privacy	Feelings of safety	Adequate security	Order received	On time delivery	Order accurate	Willingness to respond	Desire to fix issues	Promptness of service
Time-boxed releases	*0.434*	0.639	0.668	0.867	*0.630*	0.904	0.904	0.668	*0.462*	*0.462*	*0.462*	*0.496*	*0.496*	*0.496*
Operational releases	*-0.033*	*-0.225*	*-0.275*	*-0.211*	*-0.288*	*-0.044*	*-0.044*	*-0.325*	*-0.082*	*-0.082*	*-0.082*	*-0.229*	*-0.229*	*-0.229*
Small releases	*0.229*	*0.393*	*0.135*	*0.448*	*0.411*	*0.623*	*0.623*	*0.126*	*0.224*	*0.224*	*0.224*	*0.324*	*0.324*	*0.324*
Frequent releases	*0.086*	*0.070*	*-0.239*	*0.034*	*0.018*	*0.303*	*0.303*	*-0.233*	*0.018*	*0.018*	*0.018*	*0.050*	*0.050*	*0.050*
Numerous releases	*0.220*	*0.122*	*0.073*	*0.349*	0.715	*0.318*	*0.318*	*0.176*	*-0.151*	*-0.151*	*-0.151*	*0.039*	*0.039*	*0.039*
Feedback solicited	*0.456*	*0.203*	*-0.047*	*-0.054*	*-0.096*	*0.136*	*0.136*	*-0.078*	*-0.029*	*-0.029*	*-0.029*	*0.290*	*0.290*	*0.290*
Feedback received	*0.055*	*-0.240*	*-0.265*	*-0.413*	*-0.322*	*-0.220*	*-0.220*	*-0.253*	*-0.047*	*-0.047*	*-0.047*	*-0.008*	*-0.008*	*-0.008*
Feedback frequency	*0.286*	*0.132*	*-0.026*	*-0.038*	*-0.271*	*0.191*	*0.191*	*-0.147*	*0.353*	*0.353*	*0.353*	*0.194*	*0.194*	*0.194*
Feedback quality	*-0.179*	*-0.139*	*-0.364*	*-0.277*	*-0.161*	*-0.038*	*-0.038*	*-0.334*	*-0.021*	*-0.021*	*-0.021*	*0.019*	*0.019*	*0.019*
Feedback incorporated	*0.131*	*0.270*	*-0.008*	*-0.077*	*-0.298*	*0.043*	*0.043*	*-0.177*	*0.496*	*0.496*	*0.496*	*0.337*	*0.337*	*0.337*
Team leader	*0.146*	*-0.135*	*-0.161*	*-0.116*	*-0.158*	*0.039*	*0.039*	*-0.195*	*0.055*	*0.055*	*0.055*	*-0.128*	*-0.128*	*-0.128*
Vision and strategy	*0.084*	*-0.298*	*-0.231*	*-0.465*	*-0.311*	*-0.336*	*-0.336*	*-0.203*	*-0.157*	*-0.157*	*-0.157*	*-0.038*	*-0.038*	*-0.038*
Goals and objectives	*0.178*	*-0.219*	*-0.262*	*-0.377*	*-0.241*	*-0.238*	*-0.238*	*-0.274*	*-0.135*	*-0.135*	*-0.135*	*-0.052*	*-0.052*	*-0.052*
Schedules and timelines	*0.521*	*0.338*	*0.367*	*0.185*	*-0.018*	*0.311*	*0.311*	*0.290*	*0.360*	*0.360*	*0.360*	*0.481*	*0.481*	*0.481*
Small team size	*-0.177*	*-0.406*	*-0.511*	*-0.483*	*-0.263*	*-0.278*	*-0.278*	*-0.416*	*-0.431*	*-0.431*	*-0.431*	*-0.249*	*-0.249*	*-0.249*
Small size	*-0.464*	*-0.618*	-0.854	*-0.613*	*-0.209*	*-0.455*	*-0.455*	-0.746	-0.676	-0.676	-0.676	*-0.586*	*-0.586*	*-0.586*
Simple design	*-0.115*	*-0.343*	*-0.415*	*-0.344*	*0.022*	*-0.213*	*-0.213*	*-0.259*	*-0.520*	*-0.520*	*-0.520*	*-0.209*	*-0.209*	*-0.209*
Modular design	*-0.289*	*0.056*	*-0.168*	*0.036*	*0.049*	*0.122*	*0.122*	*-0.164*	*0.096*	*0.096*	*0.096*	*0.031*	*0.031*	*0.031*
Portable design	*-0.170*	*0.078*	*-0.258*	*0.101*	*0.275*	*0.136*	*0.136*	*-0.287*	*0.097*	*0.097*	*0.097*	*-0.049*	*-0.049*	*-0.049*
Extensible design	*0.339*	*0.192*	*0.030*	*0.330*	*0.612*	*0.395*	*0.395*	*0.118*	*-0.118*	*-0.118*	*-0.118*	*0.118*	*0.118*	*0.118*

Note. Data in red italics are not significant.

Table 34. Agile-Website Quality Data Variable Analysis (Adjusted R^2 Values)

Variable	In-depth information	Processing efficiency	Processing speed	Personali-zation	Product selection	Protection of privacy	Feelings of safety	Adequate security	Order received	On time delivery	Order accurate	Willingness to respond	Desire to fix issues	Promptness of service
Time-boxed releases	*0.086*	0.335	0.377	0.721	*0.321*	0.794	0.794	0.378	*0.115*	*0.115*	*0.115*	*0.152*	*0.152*	*0.152*
Operational releases	*-0.124*	*-0.068*	*-0.040*	*-0.075*	*-0.032*	*-0.123*	*-0.123*	*-0.007*	*-0.117*	*-0.117*	*-0.117*	*-0.066*	*-0.066*	*-0.066*
Small releases	*-0.066*	*0.048*	*-0.104*	*0.101*	*0.065*	*0.312*	*0.312*	*-0.107*	*-0.069*	*-0.069*	*-0.069*	*-0.007*	*-0.007*	*-0.007*
Frequent releases	*-0.117*	*-0.120*	*-0.061*	*-0.124*	*-0.125*	*-0.022*	*-0.022*	*-0.064*	*-0.125*	*-0.125*	*-0.125*	*-0.122*	*-0.122*	*-0.122*
Numerous releases	*-0.071*	*-0.108*	*-0.119*	*0.012*	0.450	*-0.011*	*-0.011*	*-0.090*	*-0.099*	*-0.099*	*-0.099*	*-0.123*	*-0.123*	*-0.123*
Feedback solicited	*0.109*	*-0.079*	*-0.123*	*-0.122*	*-0.115*	*-0.104*	*-0.104*	*-0.118*	*-0.124*	*-0.124*	*-0.124*	*-0.030*	*-0.030*	*-0.030*
Feedback received	*-0.122*	*-0.060*	*-0.046*	*0.067*	*-0.008*	*-0.071*	*-0.071*	*-0.053*	*-0.123*	*-0.123*	*-0.123*	*-0.125*	*-0.125*	*-0.125*
Feedback frequency	*-0.033*	*-0.105*	*-0.124*	*-0.123*	*-0.043*	*-0.084*	*-0.084*	*-0.101*	*0.016*	*0.016*	*0.016*	*-0.082*	*-0.082*	*-0.082*
Feedback quality	*-0.089*	*-0.103*	*0.024*	*-0.039*	*-0.096*	*-0.123*	*-0.123*	*0.000*	*-0.124*	*-0.124*	*-0.124*	*-0.125*	*-0.125*	*-0.125*
Feedback incorporated	*-0.123*	*-0.060*	*-0.143*	*-0.136*	*-0.042*	*-0.141*	*-0.141*	*-0.107*	*0.138*	*0.138*	*0.138*	*-0.013*	*-0.013*	*-0.013*
Team leader	*-0.101*	*-0.105*	*-0.096*	*-0.110*	*-0.097*	*-0.123*	*-0.123*	*-0.082*	*-0.122*	*-0.122*	*-0.122*	*-0.107*	*-0.107*	*-0.107*
Vision and strategy	*-0.117*	*-0.025*	*-0.065*	*0.119*	*-0.016*	*0.002*	*0.002*	*-0.079*	*-0.097*	*-0.097*	*-0.097*	*-0.123*	*-0.123*	*-0.123*
Goals and objectives	*-0.089*	*-0.071*	*-0.048*	*0.035*	*-0.060*	*-0.061*	*-0.061*	*-0.040*	*-0.104*	*-0.104*	*-0.104*	*-0.122*	*-0.122*	*-0.122*
Schedules and timelines	*0.180*	*0.003*	*0.027*	*-0.087*	*-0.125*	*-0.016*	*-0.016*	*-0.030*	*0.021*	*0.021*	*0.021*	*0.136*	*0.136*	*0.136*
Small team size	*-0.090*	*0.060*	*0.169*	*0.138*	*-0.047*	*-0.038*	*-0.038*	*0.069*	*0.084*	*0.084*	*0.084*	*-0.055*	*-0.055*	*-0.055*
Small size	*0.117*	*0.305*	0.695	*0.298*	*-0.076*	*0.108*	*0.108*	0.502	0.389	0.389	0.389	*0.261*	*0.261*	*0.261*
Simple design	*-0.110*	*0.007*	*0.069*	*0.008*	*-0.124*	*-0.074*	*-0.074*	*-0.050*	*0.179*	*0.179*	*0.179*	*-0.076*	*-0.076*	*-0.076*
Modular design	*-0.031*	*-0.121*	*-0.093*	*-0.124*	*-0.122*	*-0.108*	*-0.108*	*-0.095*	*-0.115*	*-0.115*	*-0.115*	*-0.124*	*-0.124*	*-0.124*
Portable design	*-0.093*	*-0.118*	*-0.050*	*-0.114*	*-0.040*	*-0.104*	*-0.104*	*-0.032*	*-0.114*	*-0.114*	*-0.114*	*-0.122*	*-0.122*	*-0.122*
Extensible design	*0.004*	*-0.084*	*-0.124*	*-0.003*	*0.296*	*0.050*	*0.050*	*-0.109*	*-0.109*	*-0.109*	*-0.109*	*-0.109*	*-0.109*	*-0.109*

Note. Data in red italics are not significant.

Factor analysis. An analysis of the four major factors of agile methods and the four major factors of website quality was performed (as shown in Table 35). Linear regression was used to build statistical models between each of the four factors of agile methods and the four factors of website quality. Few factors of agile methods, if any at all, proved to be related to any of the factors of website quality. There was one rare exception, which exhibited a high adjusted R^2 value between iterative development and privacy and security. There were some more high adjusted R^2 values between well-structured teams and privacy and security, flexibility and website design, and flexibility and privacy and security. These were significant at the 0.10 level, though the 0.05 level has been used as a strict cutoff to judge all correlations and statistical relationships. Had more data been collected on website quality, through self-report data, the statistical models between the factors of agile methods and website quality may have been related. The factors of agile methods were not correlated to a composite model of eTailQ either. This analysis indicates the factors of agile methods and website quality had little or no strong correlations, with the exception of iterative development and privacy and security (e.g., 86%).

Table 35. Agile-Website Quality Factor Analysis

Factor	Variable	Website design	Privacy and security	Fulfillment and reliability	Customer Service	eTailQ
Iterative development	*Adjusted R^2 value*	0.546	0.860	-0.120	-0.187	0.326
	F-value	3.163	12.053	0.807	0.716	1.872
	Significance	*0.144*	0.016	*0.599*	*0.644*	*0.282*
Customer feedback	*Adjusted R^2 value*	-0.869	-0.725	0.425	-0.495	-0.531
	F-value	0.256	0.328	2.184	0.470	0.445
	Significance	*0.912*	*0.869*	*0.276*	*0.784*	*0.799*
Well-structured teams	*Adjusted R^2 value*	0.540	0.729	-0.049	0.420	0.558
	F-value	3.115	5.840	0.915	2.301	3.272
	Significance	*0.147*	*0.056*	*0.549*	*0.220*	*0.137*
Flexibility	*Adjusted R^2 value*	0.740	0.656	0.038	0.287	0.538
	F-value	6.126	4.438	1.072	1.726	3.099
	Significance	*0.052*	*0.087*	*0.487*	*0.309*	*0.148*

Note. Data in red italics are not significant.

Model analysis. Five statistical models were constructed between the four major factors of website quality (including a composite model called eTailQ) and the four major factors of agile methods (as shown in Table 36). Two of the models, privacy and security and fulfillment and reliability as a function of iterative development, customer feedback, well-structured teams, and flexibility had high adjusted R^2 values (and were statistically significant). The composite model, eTailQ was significant at the 0.10 level, which was far above the minimum threshold for significance used in this analysis. About half of the Beta values associated with the factors of iterative development, customer feedback, well-structured teams, and flexibility were statistically significant. Only one of the models, fulfillment and reliability, had a high adjusted R^2 value, good F-value, high significance, and statistically significant Beta values. The weakest model was the customer service model, though few of the models were very strong. This analysis indicates the aggregated factors of agile methods are strongly correlated to two of the factors of website quality (67% and 84%) and aggregated factors of website quality (e.g., 54%).

Table 36. Agile-Website Quality Model Analysis

Model	Statistic	Website Design	Privacy and Security	Fulfillment Reliability	Customer Service	eTailQ
(*Model*)	*Adjusted R^2*	0.448	0.674	0.843	0.131	0.541
	F-value	2.829	5.648	13.097	1.339	3.650
	Significance	*0.142*	0.043	0.007	*0.372*	*0.094*
(*Constant*)	*Beta*	6.583	6.520	6.568	6.154	6.474
	t-value	5.516	5.760	5.812	2.364	4.834
	Significance	0.003	0.002	0.002	*0.064*	0.005
Iterative development	*Beta*	0.745	1.029	0.634	0.632	0.758
	t-value	3.071	4.470	2.761	1.193	2.783
	Significance	0.028	0.007	0.040	*0.286*	0.039
Customer feedback	*Beta*	0.249	0.398	3.389	2.306	1.395
	t-value	0.411	0.693	5.908	1.745	2.052
	Significance	*0.698*	*0.519*	0.002	*0.141*	*0.095*
Well-structured teams	*Beta*	-0.613	-0.809	-2.600	-1.791	-1.333
	t-value	-1.317	-1.832	-5.894	-1.763	-2.551
	Significance	*0.245*	*0.126*	0.002	*0.138*	*0.051*
Flexibility	*Beta*	-0.528	-0.708	-1.876	-1.348	-1.031
	t-value	-1.721	-2.435	-6.462	-2.015	-2.997
	Significance	*0.146*	*0.059*	0.001	*0.100*	0.030

Note. Data in red italics are not significant.

Summary of Data Analysis

Using the data from Table 36, an analysis of the hypotheses and sub-hypotheses was performed (as shown in Table 37). There was some evidence that iterative development was correlated to website quality, website design, privacy and security, and fulfillment and reliability at the 0.05 level. Customer feedback was correlated to website quality and fulfillment and reliability at the 0.10 level. Well structured teams were negatively correlated to website quality and fulfillment and reliability at the 0.10 level. Flexibility was negatively correlated to website quality, privacy and security, fulfillment and reliability, and customer service at the 0.10 level. However, our hypotheses were stated as positive correlations, so negative ones are viewed as failed hypotheses. We cannot put too much confidence in these results due to the small amount of data. The final analysis indicates iterative development and customer feedback are correlated to factors of website quality (e.g., there is some evidence that half of our hypotheses are true).

Table 37. Agile-Website Quality Summary Analysis

Factors	Hypothesis	β	*t-value*	*p-value*
Iterative development	H_1 Iterative development → Website quality	0.758	0.039	$p < 0.05$
	H_{1a} Iterative development → Website design	0.745	0.028	$p < 0.05$
	H_{1b} Iterative development → Privacy and security	1.029	0.007	$p < 0.05$
	H_{1c} Iterative development → Fulfillment and reliability	0.634	0.040	$p < 0.05$
	H_{1d} Iterative development → Customer service	0.632	*0.286*	$p > 0.10$
Customer feedback	H_2 Customer feedback → Website quality	1.395	0.095	$p < 0.10$
	H_{2a} Customer feedback → Website design	0.249	*0.698*	$p > 0.10$
	H_{2b} Customer feedback → Privacy and security	0.398	*0.519*	$p > 0.10$
	H_{2c} Customer feedback → Fulfillment and reliability	3.389	0.002	$p < 0.05$
	H_{2d} Customer feedback → Customer service	2.306	*0.141*	$p > 0.10$
Well-structured teams	H_3 Well-structured teams → Website quality	-1.333	0.051	$p < 0.10$
	H_{3a} Well-structured teams → Website design	-0.613	*0.245*	$p > 0.10$
	H_{3b} Well-structured teams → Privacy and security	-0.809	*0.126*	$p > 0.10$
	H_{3c} Well-structured teams → Fulfillment and reliability	-2.600	0.002	$p < 0.05$
	H_{3d} Well-structured teams → Customer service	-1.791	*0.138*	$p > 0.10$
Flexibility	H_4 Flexibility → Website quality	-1.031	0.030	$p < 0.05$
	H_{4a} Flexibility → Website design	-0.528	*0.146*	$p > 0.10$
	H_{4b} Flexibility → Privacy and security	-0.708	0.059	$p < 0.10$
	H_{4c} Flexibility → Fulfillment and reliability	-1.876	0.001	$p < 0.05$
	H_{4d} Flexibility → Customer service	-1.348	0.100	$p = 0.10$

Note. Data in red italics are not significant.

DISCUSSION, RESULTS, AND CONCLUSIONS

The purpose of this section is to present a discussion, the results, and conclusions of our analysis of the relationship between the use of agile methods and website quality. The objectives were to measure the degree to which organizations use agile methods, measure the quality of their websites, and then determine whether using agile methods improves website quality. There were four major elements of this study with respect to agile methods and website quality: (a) conceptual model, (b) survey instruments, (c) measurement data, and (d) data analysis. Our research questions and hypotheses are summarized as, "Does use of iterative development, customer feedback, well-structured teams, and flexibility improve e-commerce website quality?" Based on Table 37, use of iterative development and customer feedback improved website quality, but well-structured teams and flexibility did not (as shown in Figure 10). Finally, the conceptual model proved useful and agile methods were correlated to other benefits.

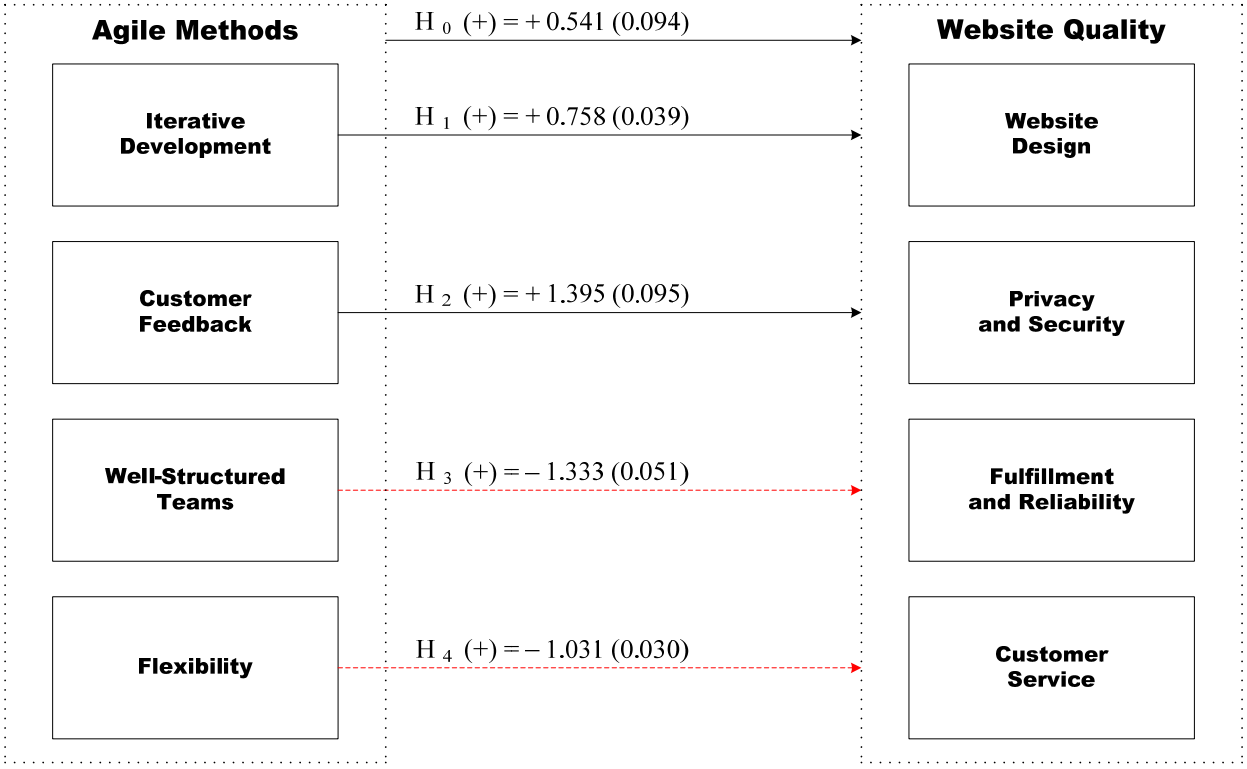

Figure 10. Final conceptual model of agile methods and website quality.

Discussion

Demographics. The summary of the demographic data from the main survey of agile methods and website quality revealed some interesting findings (as shown in Figure 11). Of the 252 respondents: (a) 250 reported their job function, (b) 252 reported their years of experience, (c) 245 reported their organization's number of employees, (d) 161 reported their organization's total annual revenue, and (e) 230 reported the industry sector of their parent organization. For job function, 80 (32%) of the people were software engineers representing the largest group of respondents. Representing senior organizational personnel, 106 (42%) of the respondents ranged from executive to chief engineer. This is both good and bad. On one hand, it may mean that agile methods are a significant topic to senior organizational personnel. On the other hand, it may mean that the use of agile methods is not that significant to technical personnel. For years of experience, 56 (22%) of the respondents had 11-15 years of experience and 118 (47%) of the respondents had more than 16 years of experience. Oftentimes, programmers only have a few years of experience, so this may mean that agile methods may not be in use by less experienced personnel. For number of employees, 115 (47%) of the respondents had less than 250 total people in their organizations. However, 15 (6%) of the respondents had more than 100,000 personnel in their organizations. This may mean that only small to medium-sized organizations are using agile methods. For annual revenue, 97 (60%) of the respondents were in organizations with less than $25 million in sales. Only 31 (19%) were in organizations earning more than $1 billion. This is more evidence that larger organizations may be shunning agile methods in favor of traditional ones. For industry sector, 143 (62%) of the respondents cited manufacturing, information, or professional as the type of parent organization from which they came. Overall, managers from small to medium-sized organizations tend to be interested in agile methods.

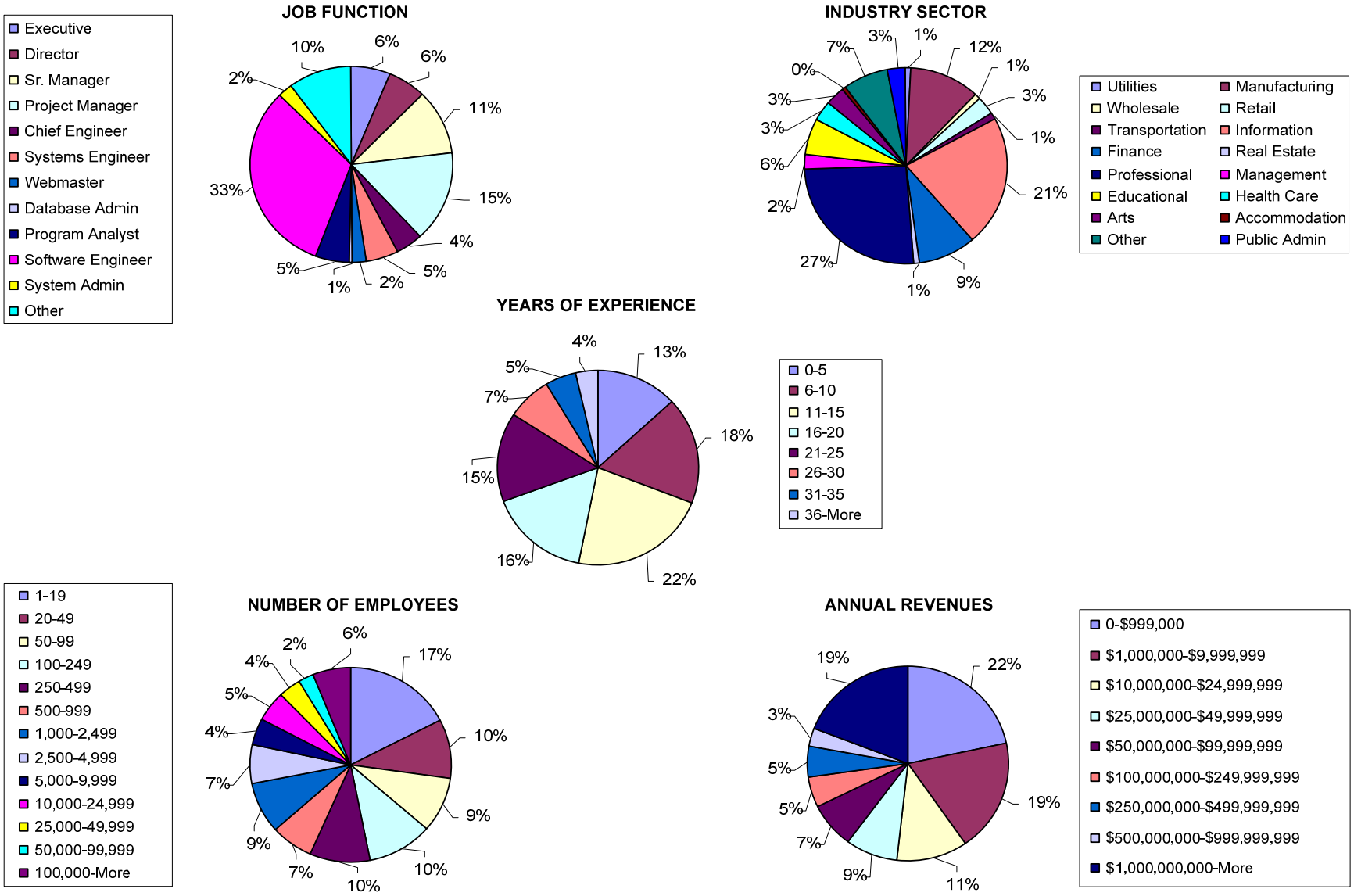

Figure 11. Summary of demographics from agile methods and website quality survey.

Agile methods. The summary of response data from the main survey of agile methods and website quality also revealed some interesting findings (as shown in Figure 12). With some exceptions, the majority of the respondents either agreed or strongly agreed with all 20 questions about agile methods. Regarding the four major factors of agile methods, iterative development and well-structured teams were the ones to which respondents most agreed and strongly agreed. At least on the surface, customer feedback and flexibility were examples of factors to which respondents least agreed. By and large, the majority of the respondents asked their customers for feedback on software iterations and fewer respondents actually received feedback from their customers. Even fewer respondents received feedback from their customers in a timely manner and only half of them received high quality feedback from their customers, but nearly all of the respondents incorporated feedback they did receive from their customers. Agile methods may or may not be helping people overcome these systemic issues to traditional software development methods with regard to getting customer feedback. For well-structured teams, the majority of the respondents agreed with all statements, but strongly agreed with small team size. Small team size may simply have evolved into a cliché within the software industry, and the magnitude of the response rate to small team size skewed the analysis of the data in a negative way. However, many software teams fail from having more personnel than necessary, so small team size should not be removed from the survey instrument or data. Only 6% of the respondents came from organizations in excess of 100,000 people, so maybe small team size can be removed as a constraint for small organizations (since they use small teams by default). One argument is that the survey instrument accurately gauges use of agile methods, while bias toward the higher end of the scales may also be argued. Low correlations between agile methods, benefits, and website quality may indicate the presence of bias by the respondents to answer positively.

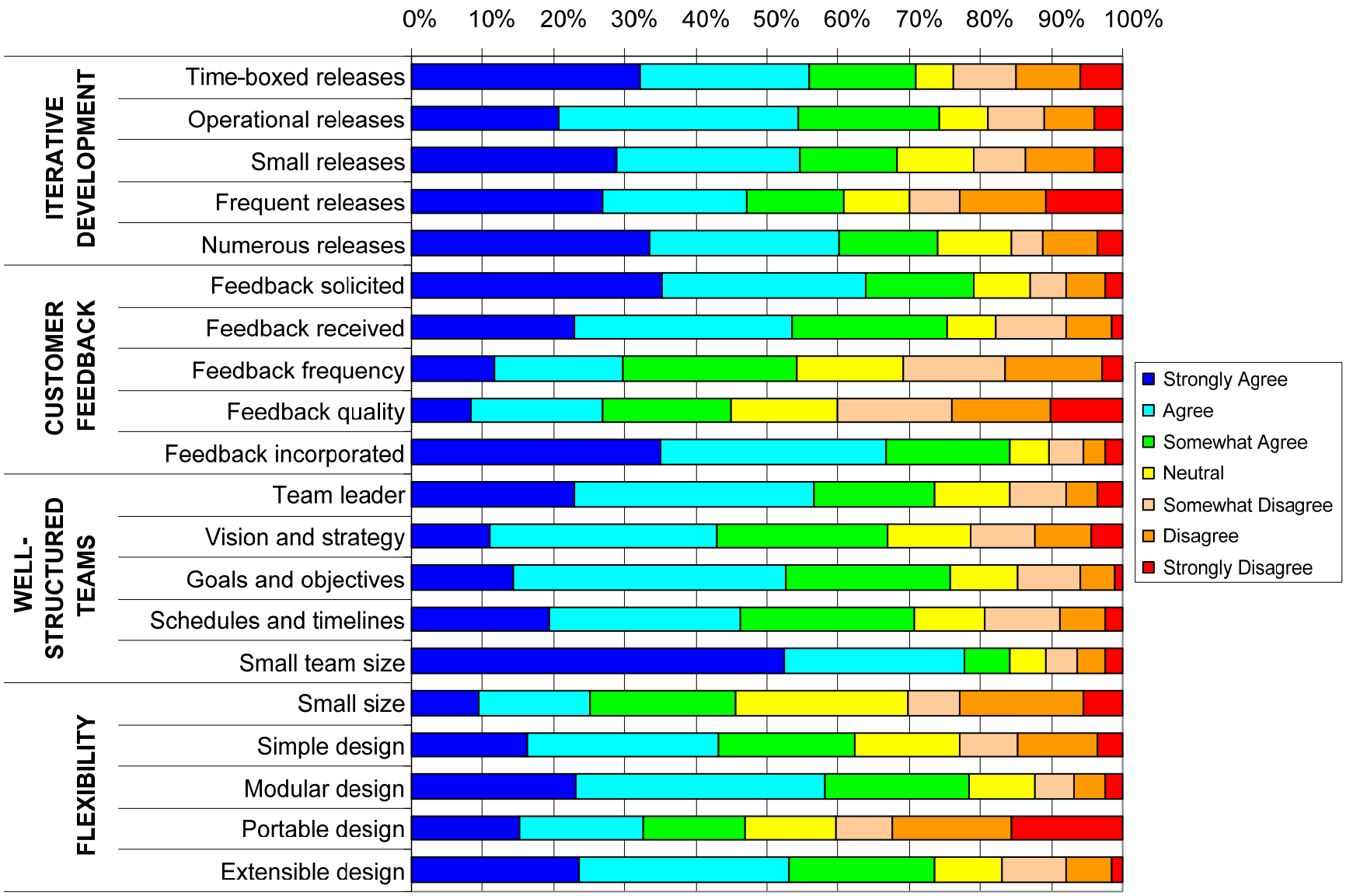

Figure 12. Summary of responses from agile methods survey.

Benefits. The summary of benefit data from the main survey of agile methods and website quality also revealed some interesting findings (as shown in Figure 13). Self-reported benefit data were collected to help gauge the effects of using agile methods in the interim period before the final website data were collected and correlated to agile methods. Only 138 (55%) of the respondents supplied data about the benefits of using agile methods, which is significant in of itself. This may indicate that 45% of the respondents were not confident enough to make assertions about the benefits of agile methods. Some argue that self-reported data is biased by nature, while others rely principally upon self-reported data when analyzing the costs and benefits of software methods. The estimates of cost efficiency, productivity, quality, and cycle-time collected by this study were relatively conservative and tended to be in the lower ranges. As an exception, the largest group of respondents stated that customer satisfaction improved with the use of agile methods by 76% to 100%. The median score for all self-reported benefit data was 3.44, which indicates that the majority of respondents felt agile methods did not increase the value of any one category of benefits by more than 50%. As we saw earlier, the responses towards the main survey of agile methods were biased towards the upper end of the scales, while responses to self-reported benefit data were skewed toward the lower end of the scales. This resulted in some interesting correlations between agile methods and self-reported benefit data. Cost efficiency and productivity were the least reported benefits of agile methods, and quality, cycle-time, and customer satisfaction were the most often cited benefits of agile methods. There were good correlations between all of the benefit data, meaning the data were evenly reported. Only cost and quality were significantly correlated to all of the factors of agile methods. While it was useful to collect self-reported benefit data in the interim period when the study was in-progress, it would have been useful to collect self-reported data about website quality as well.

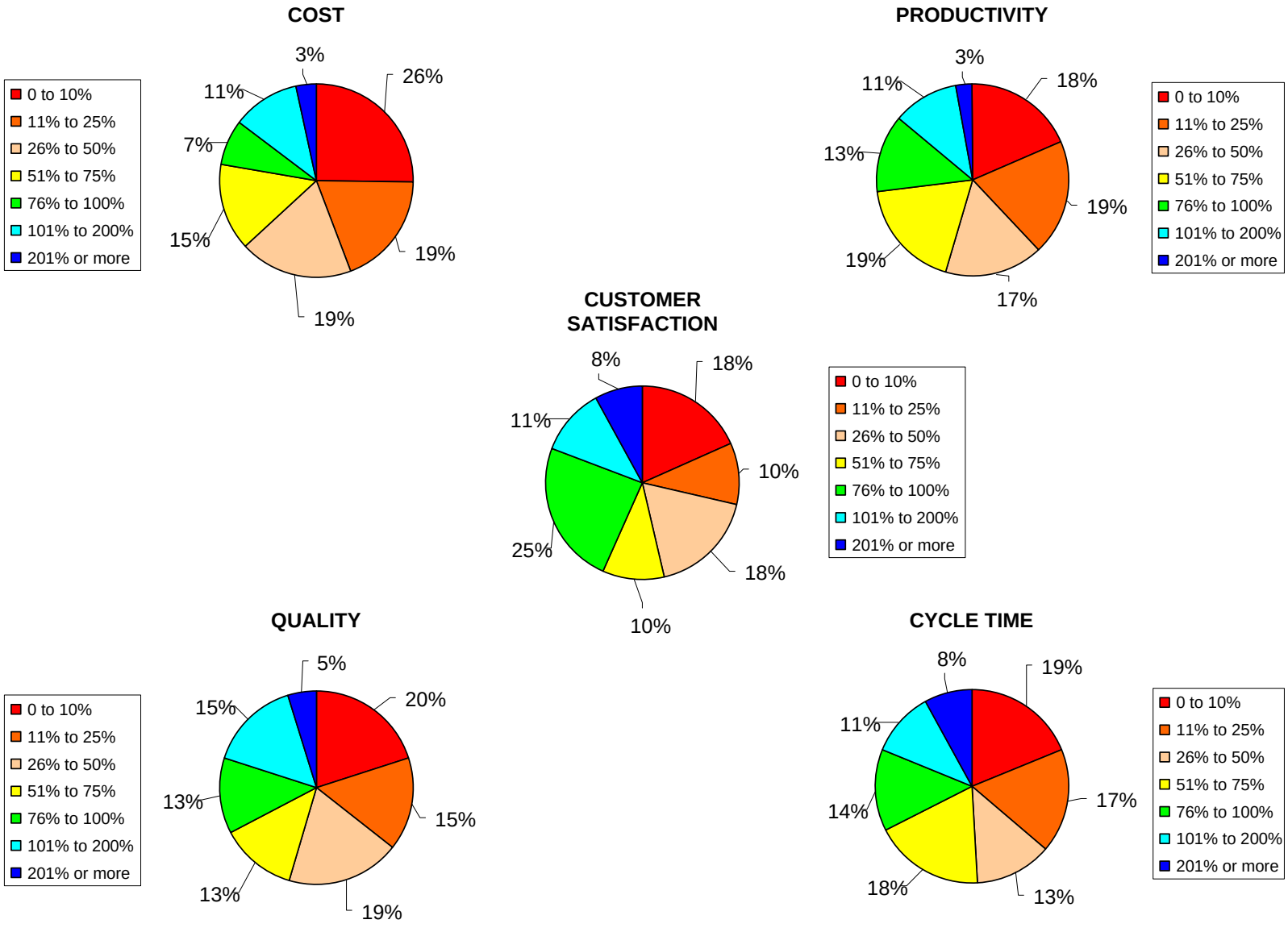

Figure 13. Summary of responses from benefit survey.

Website quality. The summary of website quality data from the main survey of agile methods and website quality was simple, but interesting (as shown in Figure 14). The 14-point eTailQ instrument was used to assess the quality of websites supplied by the respondents to the main survey of agile methods. Approximately 27 respondents supplied Internet addresses of websites for further quality analysis. Of these, 10 of the Internet addresses met the criteria for being classified as e-commerce websites, versus informational websites. A product was ordered from each of the 10 e-commerce Internet addresses and the eTailQ instrument was used to judge the completed end-to-end transaction (since that is what eTailQ was designed to measure). As the transactions were transpiring, the data with respect to website design, privacy and security, fulfillment and reliability, and customer services was supplied. Only after the products were received was this phase of the study considered complete. Then, an analysis of the website quality data ensued, including a correlational analysis to agile methods. The majority of ratings for website design were in the agree range. The ratings for privacy and security were in the strongly agree range. The ratings for fulfillment and reliability were in the agree to strongly agree range. The ratings of customer service were in the strongly agree range. Note that all of the responses were skewed toward the agree to strongly agree range. This was also true of the agile methods data. In fact, the average score of responses to agile methods was about 5, whereas the average score for website quality was 5.7. So the data with regard to agile methods and website quality seemed to be skewed in the same direction. The eTailQ instrument itself has proven to be reliable under repeated field testing on large numbers of respondents. From this discussion, we conclude that these were reasonably accurate ratings of the 10 e-commerce websites resulting from the main survey of agile methods. The only obvious weakness is the low volume of data (e.g., 10), which could have been averted if self-reported data were collected.

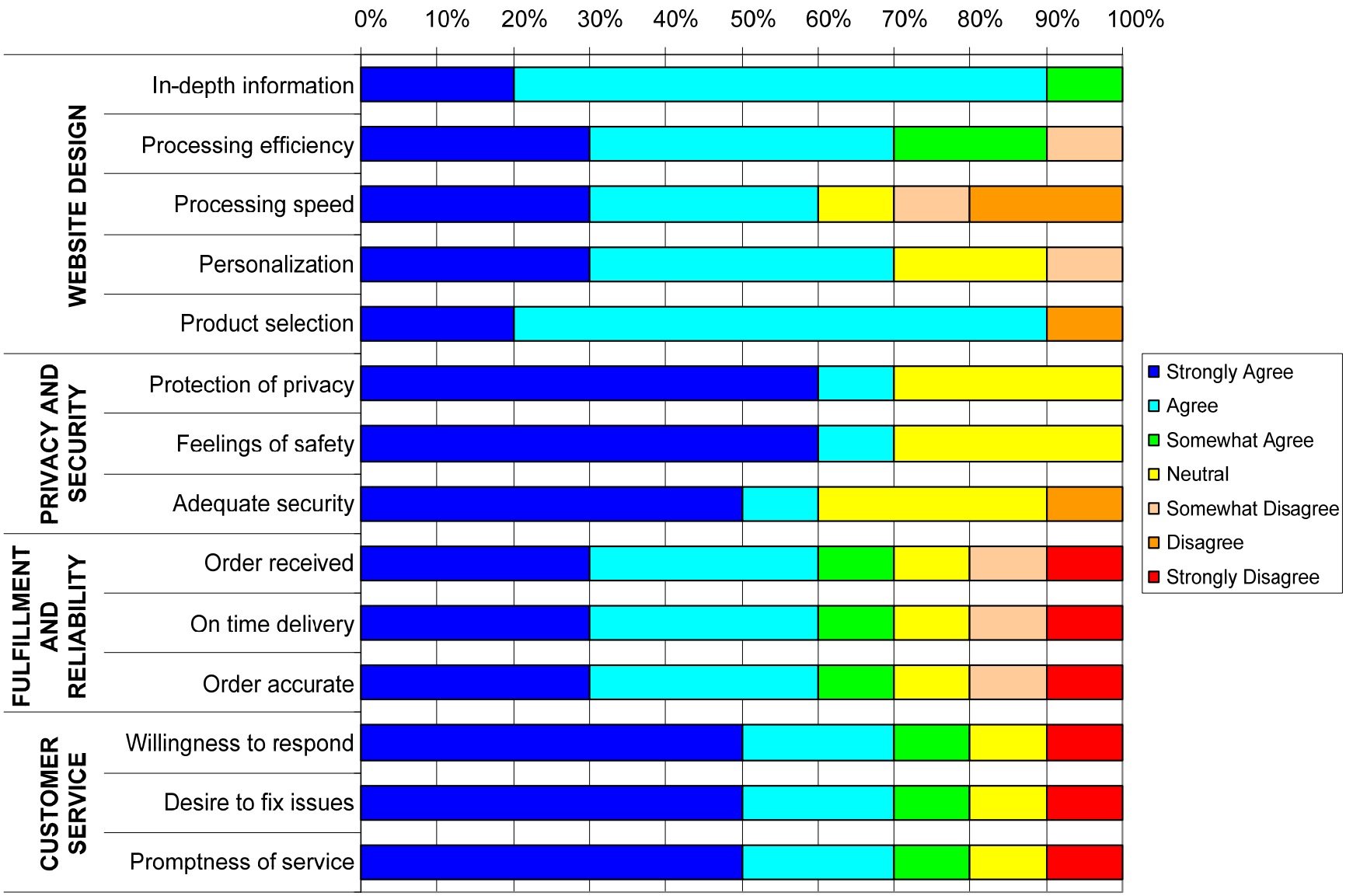

Figure 14. Summary of responses from website quality assessment.

Results

Benefits. Based on Table 32, the relationships between agile methods and website quality showed some interesting and significant findings (as shown in Figure 15). To provide some context, very good relationships would be in the 0.800 or 0.900 range, which would mean the beta values were correlated and had good explanatory power. The strongest relationship is between iterative development and cycle-time. The weakest relationship is between customer feedback and quality. (Note that the numbers in parentheses represent statistical significance and numbers in red italics are insignificant, e.g., higher numbers in parentheses are less significant. The significant relationships are mostly at the 0.05 level, while some are at the 0.10 level.) While individual correlations between the factors and variables of agile methods and benefits were good, grouped relationships like these tend to point out inconsistencies. For instance, the reported benefits between the four factors of agile methods were not consistent.

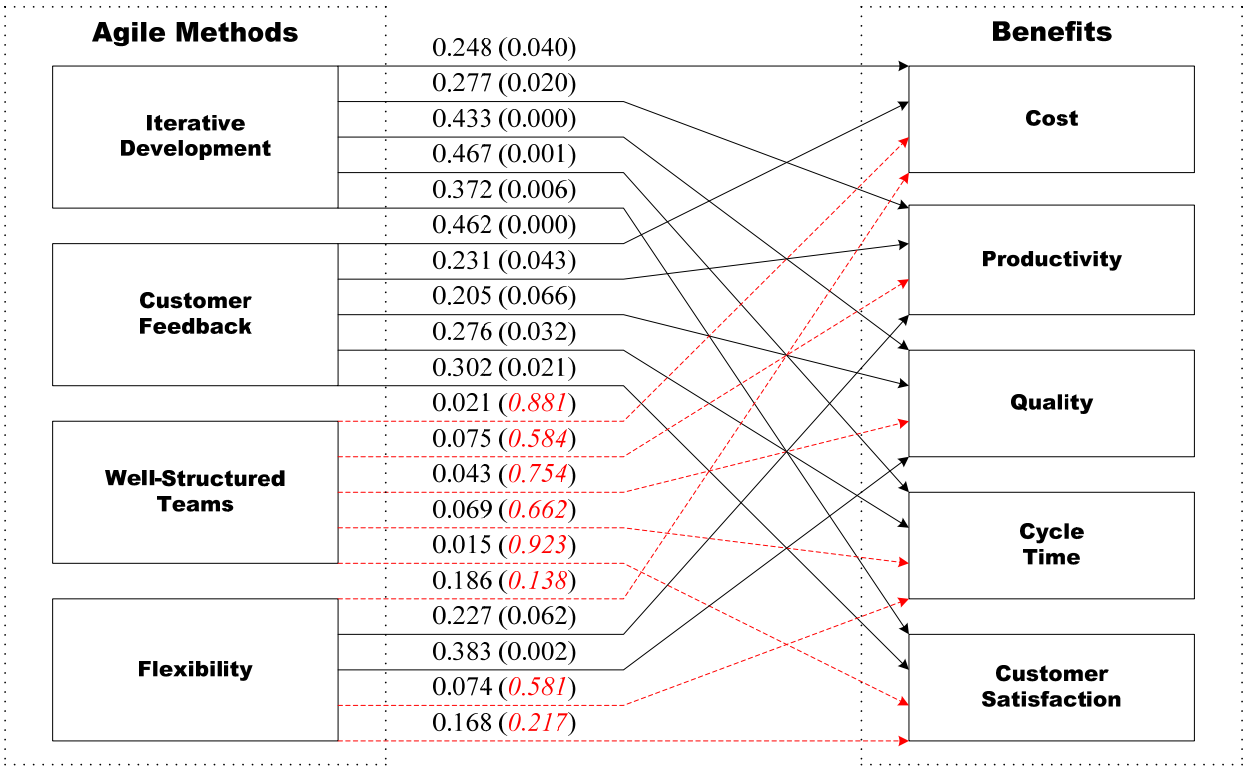

Figure 15. Summary of relationships between agile methods and benefits.

Website Quality. Based on Table 37, the sub-hypotheses associated with the conceptual model further depict the relationships between the factors of agile methods and website quality (as shown in Figure 16). Use of iterative development was related to improved website design, privacy and security, and fulfillment and reliability. Customer feedback was related to improved fulfillment and reliability. However, the majority of the sub-hypotheses between customer feedback, well-structured teams, flexibility, and the four major factors of website quality were either non-existent or negative. While data was collected from 250 respondents on agile methods, data was only available on 10 websites, so we can't put too much emphasis on either the positive or the negative correlations. More data on websites would increase our confidence in the results, but would not guarantee any more positive or negative correlations between the factors of agile methods and website quality. (Note that the data shown in red italics indicate little or no statistical significance.)

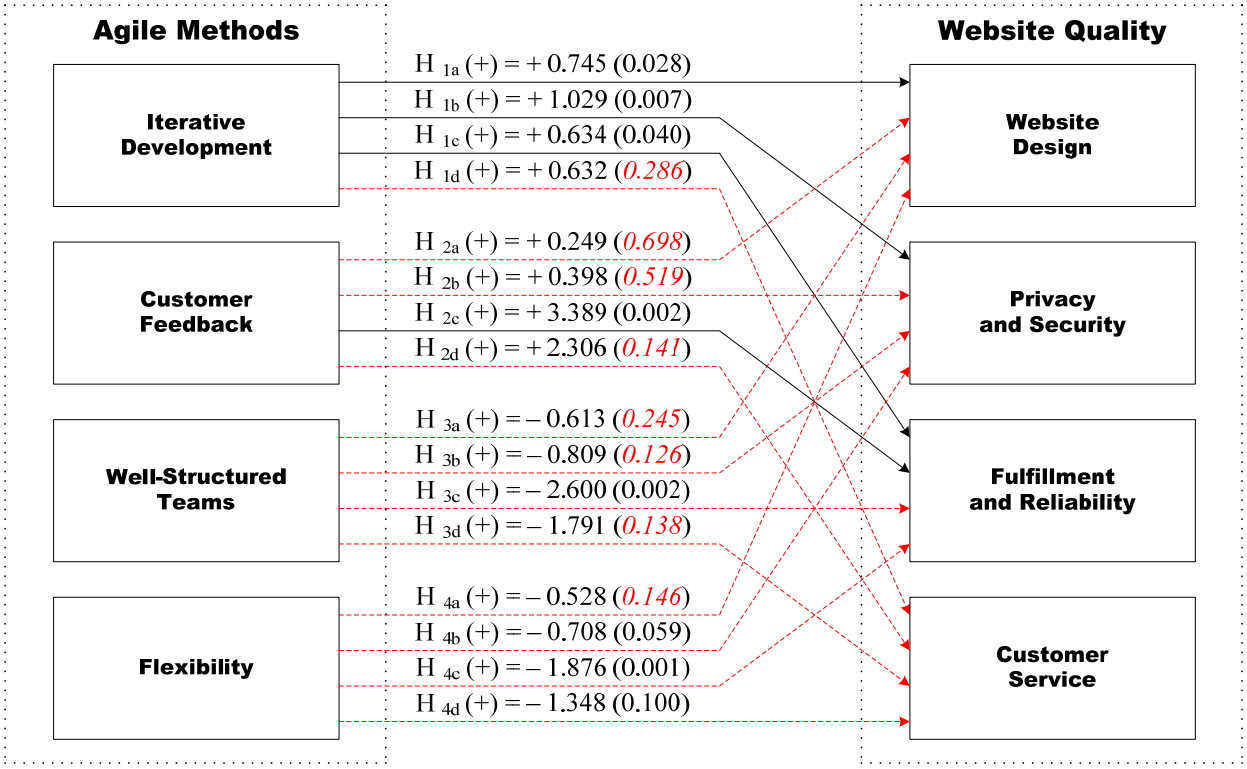

Figure 16. Summary of relationships between agile methods and website quality.

Limitations

There were several limitations associated with this study: (a) use of a new conceptual model of agile methods, (b) use of a new survey instrument of agile methods, (c) use of self-selected respondents, and (d) use of a small number of websites for correlation to agile methods. First, our conceptual model was custom-designed for this study, and it will require further trial testing to determine its reliability and validity before it is complete. Second, our agile methods survey instrument was also custom-designed for this study, and it will require further trial testing and analysis to determine the extent to which it is reliable and valid. Third, it is difficult to measure the degree to which the use of self-selected respondents biased our responses about agile methods, because they may have been proponents of agile methods. Fourth, we definitely needed a larger sample of websites for our analysis to be meaningful, which may have been possible if we had collected self-reported data from a larger industry sector. Lastly, we did not analyze the quality of websites using traditional methods, and we omitted speed-of-delivery from the conceptual model, which may be a major benefit of agile methods.

Lessons

There were several lessons we learned from our study, which could help other scholars with similar studies. First, choose a larger and slower industry to study and use a general-purpose model of software quality to maximize the amount of data one can possibly obtain. Second, use cognitive interviews to pre-test surveys, interview novices and experts alike, and conduct trial-runs and pilot surveys to evaluate your survey instruments early. Fourth, use online survey websites to collect data, especially ones that are inexpensive, flexible, and easy-to-use, rather than conducting phone, snail-mail, email, or traditional paper surveys. Fifth, use popular web blogs to promote surveys instead of email surveys, of which good ones are hard to find. Sixth, keep your conceptual models and survey instruments as small as possible. Seventh, collect self-reported checkpoint data as interim stop-gap measures and provide incentives to respondents. Lastly, try to find a regular forum or blog for conducting multiple shorter data collection cycles.

Contributions

There were several major contributions of this study. First, we developed a new conceptual model of agile methods. Second, we developed a general-purpose survey instrument to measure the use of agile methods. (These are considered both contributions and limitations, because they are novel yet untested.) Third, we collected original measurements from 250 respondents on the use of agile methods, which may be useful to other scholars. Fourth, we identified best-of-breed models for measuring the quality of e-commerce websites, which was no easy task considering over 50 scholarly models have been devised in the last 10 years alone. Fifth, we presented our conceptual model at a major academic conference and vetted our preliminary design. Sixth, we published a short article summarizing our initial survey results in a British journal specializing in information systems quality issues. Seventh, our literature review has been selected to be published as a chapter in a scholarly textbook on computer science.

Conclusions

Does use of agile methods improve the quality of Internet websites used for the $2 trillion electronic commerce industry by U.S. firms? Does the use of iterative development, customer feedback, well-structured teams, and flexibility lead to higher quality Internet websites? A survey of 250 respondents was conducted to help determine whether the use of agile methods is linked to website quality. The results of this study showed that the use of iterative development and customer feedback is linked to higher website quality, but not well-structured teams and flexibility. However, agile methods were linked to other benefits such as improvements in cost efficiency, productivity, quality, cycle time, and customer satisfaction. Other contributions of this study include a history of agile methods, a conceptual framework, survey instruments, a repository of original data, and a roadmap for conducting future studies of agile methods.

Conceptual model. A goal of this study was to identify or develop a conceptual model of agile methods to measure the relationships between use of agile methods and website quality. There are many types and kinds of agile methods in-use, so one of our challenges was to identify or develop a conceptual model, which could measure the common properties of agile methods. Our first preference would have been to use a conceptual model with proven properties of reliability and validity, which could avert the risks of developing a new model of agile methods. Failing that, we were left with the task of analyzing the properties of agile methods, devising a new conceptual model of agile methods, and then using it to collect original measurement data. Our new conceptual model of agile methods seemed to have proven useful for measuring the common properties of multiple agile methods, based on measurements from 250 respondents.

Benefit data. Benefits were collected as interim measures to gauge the effects of agile methods apart from the main goal of studying links between agile methods and website quality. Little is known about the costs and benefits of using agile methods to develop software products, which precipitated this study of the relationships involving agile methods and website quality. There were strong correlations between the benefit variables, and composite statistical models also reinforced the existence of these relationships, which indicates the benefit data were reliable. Furthermore, there were correlations between the use of agile methods and the benefit data, which indicates both the conceptual model of agile methods and the benefit data may be reliable. Perhaps, a more sophisticated model of costs and benefits could have been used to test and reinforce the existence of the relationships between agile methods and benefit data.

Website quality. Another goal of this study was to identify or develop an instrument for accurately gauging the quality of websites and then relate their quality to use of agile methods. Fortunately, an analysis of literature on website quality revealed the existence of nearly 50

scholarly models of website quality produced in the last 10 years, which simplified this aspect. Our challenge was to sort through the models and make a determination about the one best suited for measuring website quality, which could easily be integrated with a model of agile methods. The eTailQ model of website quality seemed suitable for our purposes, since its theoretical tenets were based on measuring the entire e-commerce transaction and it was small and very reliable. An analysis of the website quality data collected as part of this study indicates that the eTailQ instrument was a suitable choice for measuring website quality.

Agile methods and website quality. The fundamental purpose of this study was to help determine whether using agile methods to produce e-commerce websites improves their quality. Using our conceptual model of agile methods, we collected measurement data from over 250 respondents, and analysis of this data indicated it had reasonably good reliability and validity. Furthermore, our 250 respondents volunteered Internet addresses of 27 websites they created using agile methods, 10 of which met the criteria for being classified as e-commerce websites. Using the eTailQ model of website quality, we assessed the quality of the 10 websites based on completed end-to-end transactions, and analysis of this data was reasonably reliable and valid. Extensive analysis of these data using linear regression revealed weak relationships at best between the use of agile methods and website quality measured using the eTailQ instrument.

E-commerce industry. The $2 trillion U.S. e-commerce industry was the context and backdrop of this study of the relationships between the use of agile methods and website quality. Today, there are over 100 million websites and $130 billion of these revenues come from what is known as the business-to-consumer sector, such as online shopping websites like Amazon, Inc. So one of the fundamental assumptions of this study was that a reasonable amount of our respondents would be developing e-commerce websites, which may not be a good assumption.

Only 10 out of 250 respondents reported the Internet addresses of e-commerce websites they produced, which indicates that only 4% of our respondents are producing e-commerce websites. If this is correct, then 96% of our respondents are producing software products other than e-commerce websites, and a model of software quality should have been used suitable to them. Perhaps measuring the quality of commercial, shrink-wrapped software is more appropriate.

Research method. We used a two-phase, quasi-longitudinal study to measure the relationships between use of agile methods to produce e-commerce websites and their quality. This was a reasonable research design to help ensure the reliability and validity of our data (e.g., an independent assessment of the quality of websites made by respondents using agile methods). Pilot surveys indicated that our respondents would report a low number of Internet addresses, precipitating our decision to collect stop-gap, checkpoint data on the benefits of agile methods. In hindsight, we should have continued with our two-phase, quasi-longitudinal study of agile methods and website quality, but collected self-reported data on website quality using eTailQ. This may not have solved our problem of obtaining such a small number of Internet addresses, but it may have yielded more useful correlations between agile methods and website quality.

Data analysis. Our principal method of data analysis for examining the relationships between agile methods and website quality was correlational analysis and linear regression. The large volume of data made it prohibitive to continue looking for curvilinear, log-linear, and quadratic relationships between the data, and our analysis did not indicate this would be fruitful. Furthermore, we did not perform a factor analysis of the data by adding or removing variables and factors to look for the best possible-fit between the agile methods and website quality data. In fact, a factor analysis would have been contrary to the fundamental goals and objectives of this study, which were to analyze the factors of agile methods *in toto*, rather than a best-fit

model. However, the analysis we did perform was rather thorough, and we did find many relationships within and between the data of agile methods, their benefits, and website quality.

Final Summary

The purpose of this study was to help determine whether the use of agile methods to develop e-commerce websites improved their quality. Our study revealed some relationships between the use of agile methods and website quality based on data from 250 respondents and 10 websites. However, the relationships were weak and it would require more data to substantiate these relationships and find stronger correlations between agile methods and website quality. We also developed a general-purpose conceptual model of agile methods, an original survey instrument, and we collected original data on the use of agile methods worthy of further study. Finally, we collected stop-gap measures of benefits which were correlated to agile methods.

RECOMMENDATIONS FOR FUTURE WORK

The purpose of this section is present recommendations for future work with regard to the study of relationships between the use of agile methods and the quality of e-commerce websites. We conducted a textbook study formulating a research problem, conducting a literature review, designing a conceptual model, identifying a research method, and collecting and analyzing data. We did some things right, some things wrong, and we learned a lot of valuable lessons about how to improve the outcomes of scholarly research, including compressing the research cycle. Recommendations for future work include studying a broader industry sector, refining the conceptual model, scaling the research scope, and carefully selecting proper outcome measures. More recommendations include reexamining our research method, carefully selecting our source of data, and ensuring the success of our data collection. Some of these recommendations are short-term improvements and some are long-term improvements.

Industry Sector

Initially, we chose to study the $2 trillion U.S. e-commerce industry. Experience has shown that this is a very narrow industry sector and data collection is proved very challenging. At first glance, a broader industry sector seems more appropriate, such as the U.S. government or military sector, since the U.S. spends $400 billion on defense and employs millions of workers. However, it may be best to broaden the industry sector to an entire category such as information technology, engineering, or manufacturing, or even all three. Or, perhaps it would be better to study the factors of information system outcomes common to as many industry sectors as possible. For instance, the title of this research study could have been posited like this, “The Effects of Agile Methods on Information Systems Quality within the U.S. Marketplace.” In doing so, we increase the population of respondents to this study and the likelihood of success.

Research Scope

The purpose of this study was to examine the outcomes of the factors of agile methods, such as using iterative development, customer feedback, well-structured teams, and flexibility. Recent literature reinforces the value of these principles, especially early prototypes, market focus, cross-functional teams, and flexibility of policies, processes, and manufacturing materials. The classical formula for successful technology management includes people, process, and technology, of which agile methods embodies all three. Agile methods imply that use of talented personnel outweighs the process, so perhaps we need to reexamine the quality of personnel. What this implies is that we may want to broaden the scope of future studies of agile methods to include more organizational-level issues. Whereas agile methods tend to be associated with the use of small programming teams, team sizes of several hundred people have been successful even with flexible new product development processes similar to those of agile methods.

Conceptual Model

Another recommendation for future work would be to refine the conceptual model. More study is needed to identify the optimal subfactors of agile methods, especially well-structured teams and flexibility. Another tactic would be to focus the research and attempt four smaller studies to pin down the subfactors of iterative development, customer feedback, well-structured teams, and flexibility. There is a great need to validate individual conceptual models of iterative development, customer feedback, well-structured teams, and flexibility. In some cases, iterative development is too slow and in some cases it is too fast. What is the optimal pace for iterative development? What are the issues of customer feedback? How large can teams become before they stop being agile? What does software flexibility really mean? Is speed-of-delivery the major benefit of agile methods (versus traditional measures such as information systems quality)?

Outcome Measures

Our goal was to study the use agile methods for producing websites, so we latched on to the notion of measuring website quality and we had 50 scholarly models from which to choose. Another recommendation would be to broaden the scope from measuring website quality to the identification of a small and reliable instrument for measuring information systems quality. Remember that our survey of agile methods yielded 250 respondents and the Internet addresses of only 27 websites. What if we could have chosen a scope that applied to all 250 respondents? That would be the goal of a future study—Identify or develop a general-purpose model for measuring information systems quality. Our study also resulted in data relating to the benefits of using agile methods. This data was correlated to the use of agile methods. Another tactic would be to study other information systems outcomes, such as the benefits of agile methods, including customer satisfaction, which is commonly associated with the use of agile methods.

Research Method

We used survey research to collect our data, because we were interested in quantitative outcomes and we wanted to collect a larger sample of data from which to draw conclusions. Another recommendation would be to use qualitative research methods such as case studies, interviews, and other small scale methods, which would yield less quantitative, but richer results. There is evidence that information system developers do not use any method at all. Perhaps, qualitative research methods would yield new conclusions about the way software is developed. Another recommendation would be experimental research. That is, conduct a small scale experiment to study of the effects and outcomes of using agile methods to develop software. Yet another recommendation would be to completely reexamine the best way to collect data. Perhaps, we can think of ways to achieve our objectives by analyzing existing data.

Data Source

We utilized survey research as our primary data collection method, but the source of our data came about rather haphazardly. One recommendation would be to carefully locate multiple sources of data far in advance of data collection. For instance, we were able to yield data from 250 respondents by having our survey instrument mentioned on a popular Internet blog. Perhaps, we need to identify the top 10 Internet blogs and figure out how to tap into their potential for conducting information systems research. Another source of data would be to secure the commitment of a small number of organizations for analysis. Yet another data source would be to survey the customers of organizations using agile methods. Perhaps, we can identify the top 50 e-commerce websites and then determine how to gain access to their developers and survey their use of agile methods. Perhaps avenues will open up in the future to survey certified web masters or personnel certified in the use of agile methods.

Data Collection

Lastly, we endeavored to conduct a two-phase study of agile methods and website quality. That is, we intended to survey the practices of software developers. Then, we intended to analyze the quality of their websites. This seemed like a logical data collection strategy maintain the objectivity of the study. However, it is important to collect checkpoint measures to ensure outcome data are collected just in case the second phase of data collection does not yield useful results. We collected checkpoint data on the benefits of agile methods, which proved very useful. Therefore, we recommend that self-reported checkpoint data be collected on the outcome measures of your choice. This procedure would help ensure that data are collected, which may be useful for studying the relationships between the dependent and independent variables, regardless of what happens during the second phase of the study.

REFERENCES

Abdel-Hamid, T. K. (1988). The economics of software quality assurance: A simulation based case study. *MIS Quarterly, 12*(3), 394-411.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis* (478). Oulu, Finland: VTT Publications.

Accredited Standards Committee. (2006). *The creation of ASC X12*. Retrieved October 22, 2006, from http://www.x12.org/x12org/about/X12History.cfm

Adrion, W. R., Branstad, M. A., & Cherniavsky, J. C. (1982). Validation, verification, and testing of computer software. *ACM Computing Surveys, 14*(2), 159-192.

Agarwal, R., Prasad, J., Tanniru, M., & Lynch, J. (2000). Risks of rapid application development. *Communications of the ACM, 43*(11), 177-188.

Agile Manifesto. (2001). *Manifesto for agile software development*. Retrieved November 29, 2006, from http://www.agilemanifesto.org

Akiyama, F. (1971). An example of software system debugging. *Proceedings of the International Federation for Information Processing Congress, Ljubljana, Yugoslavia,* 353–379.

Alavi, M. (1985). Some thoughts on quality issues of end-user developed systems. *Proceedings of the 21st Annual Conference on Computer Personnel Research, Minneapolis, Minnesota, USA,* 200-207.

Albrecht, A. J. (1979). Measuring application development productivity. *Proceedings of the IBM Applications Development Joint SHARE/GUIDE Symposium, Monterrey, California, USA,* 83-92.

Allen, T. J. (1966). Studies of the problem solving process in engineering design. *IEEE Transactions on Engineering Management, 13*(2), 72-83.

Ambler, S. W. (2006). *Agile adoption rate survey: March 2006*. Retrieved September 17, 2006, from http://www.ambysoft.com/downloads/surveys/AgileAdoptionRates.ppt

American National Standards Institute. (1988). *Information resource dictionary system* (ANSI X3.138-1988). New York, NY: Author.

Amey, W. W. (1979). The computer assisted software engineering (CASE) system. *Proceedings of the Fourth International Conference on Software Engineering, Munich, Germany,* 111-115.

Anderson, A., Beattie, R., Beck, K., Bryant, D., DeArment, M., Fowler, M., et al. (1998). Chrysler goes to extremes. *Distributed Computing Magazine, 1*(10), 24-28.

Anderson, R. M. (1966). Management controls for effective and profitable use of EDP resources. *Proceedings of the 21st National Conference for the Association for Computing Machinery, New York, NY, USA*, 201-207.

Anonymous. (1965). A fascinating teller. *Banking, 58*(3), 95-95.

Anonymous. (1971). Tela-fax takes dead aim on checks, credit cards. *Banking, 64*(3), 52-53.

Anonymous. (1975). Bank patents its EFT system. *Banking, 67*(1), 88-88.

Arambewela, R., & Hall, J. (2006). A comparative analysis of international education satisfaction using servqual [Special issue]. *Journal of Services Research, 6*, 141-163.

Arango, G. (1988). *Domain engineering for software reuse* (ICS-RTP-88-27). Irvine, CA: University of California Irvine, Department of Information and Computer Science.

Arthur, L. J. (1985). *Measuring programmer productivity and software quality*. New York, NY: John Wiley & Sons.

Athaide, G. A., Stump, R. L., & Joshi, A. W. (2003). Understanding new product co-development relationships in technology based industrial markets. *Journal of Marketing Theory and Practice, 11*(3), 46-58.

Augustine, S. (2005). *Managing agile projects*. Upper Saddle River, NJ: Prentice Hall.

Bachman, C. W. (1969). Data structure diagrams. *ACM SIGMIS Database, 1*(2), 4-10.

Bailey, J. E., & Pearson, S. W. (1983). Development of a tool for measuring and analyzing computer user satisfaction. *Management Science, 29*(5), 530-545.

Baker, F. T. (1975). Structured programming in a production programming environment. *Proceedings of the First International Conference on Reliable Software, Los Angeles, California, USA*, 172-185.

Banker, R. D., & Kauffman, R. J. (1991). Reuse and productivity in integrated computer aided software engineering: An empirical study. *MIS Quarterly, 15*(3), 375-401.

Barki, H., & Hartwick, J. (1994a). User participation, conflict, and conflict resolution: The mediating roles of influence. *Information Systems Research, 5*(4), 422-438.

Barki, H., & Hartwick, J. (1994b). Measuring user participation, user involvement, and user attitude. *MIS Quarterly, 18*(1), 59-82.

Barki, H., & Hartwick, J. (2001). Interpersonal conflict and its management in information systems development. *MIS Quarterly, 25*(2), 195-228.

Barnes, S. J., & Vidgen, R. T. (2000). Webqual: An exploration of web site quality. *Proceedings of the Eighth European Conference on Information Systems, Vienna, Austria*, 298-305.

Barnes, S. J., & Vidgen, R. T. (2001). An evaluation of cyber bookshops: The webqual method. *International Journal of Electronic Commerce, 6*(1), 11-30.

Baroudi, J. J., & Orlikowski, W. J. (1988). A short form measure of user information satisfaction: A psychometric evaluation and notes on use. *Journal of Management Information Systems, 4*(4), 44-59.

Basili, V. R., & Reiter, R. W. (1979). An investigation of human factors in software development. *IEEE Computer, 12*(12), 21-38.

Basili, V. R., & Turner, J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering, 1*(4), 390-396.

Bauer, F. L. (1976). Programming as an evolutionary process. *Proceedings of the Second International Conference on Software Engineering, San Francisco, California, USA*, 223-234.

Bauer, R. A., Collar, E., & Tang, V. (1992). *The silverlake project: Transformation at IBM*. New York, NY: Oxford University Press.

Bechtolsheim, A. (1978). Interactive specification of structured designs. *Proceedings of the 15th Annual ACM IEEE Design Automation Conference, Las Vegas, Nevada, USA*, 261-263.

Beck, K. (1999). Embracing change with extreme programming. *IEEE Computer, 32*(10), 70-77.

Beck, K. (2005). *Extreme programming: Embrace change*. Upper Saddle River, NJ: Pearson Education.

Blili, S., Raymond, L., & Rivard, S. (1998). Impact of task uncertainty, end-user involvement, and competence on the success of end-user computing. *Information and Management, 33*(3), 137-153.

Boehm, B. W. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes, 11*(4), 14-24.

Boehm, B. W., & Ross, R. (1988). Theory W software project management: A case study. *Proceedings of the 10th International Conference on Software Engineering, Singapore*, 30-40.

Boehm, B. W., Brown, J. R., Kaspar, H., & Lipow, M. (1973). *Characteristics of software quality* (TRW-SS-73-09). Redondo Beach, CA: TRW Corporation.

Booch, G. (1981). Describing software design in ada. *SIGPLAN Notices, 16*(9), 42-47.

Borck, J., & Knorr, E. (2005). A field guide to hosted apps. *Infoworld, 27*(16), 38-44.

Bratman, H., & Court, T. (1975). The software factory. *IEEE Computer, 8*(5), 28-37.

Bretthauer, D. (2002). Open source software: A history. *Information Technology and Libraries, 21*(1), 3-10.

Brown, J. R., & Lipow, M. (1975). Testing for software reliability. *Proceedings of the First International Conference on Reliable Software, Los Angeles, California, USA,* 518-527.

Buckley, A., Tse, K., Rijken, H., & Eijgenhuijsen, H. (2002). Stock market valuation with real options: Lessons from netscape. *European Management Journal, 20*(5), 512-526.

Byrd, T. A., & Turner, D. E. (2001). The exploratory examination of the relationship between flexible IT infrastructure and competitive advantage. *Information and Management, 39*(1), 41-52.

Caccese, M. S., & Lim, C. H. (2005). Revised global investment performance standards: Highlights and recommendations. *Investment Lawyer, 12*(12), 3-10.

Callahan, R. E. (1979). A management dilemma revisited: Must businesses choose between stability and adaptability? *Sloan Management Review, 21*(1), 25-33.

Campbell-Kelly, M. (1995). Development and structure of the international software industry: 1950-1990. *Business and Economic History, 24*(2), 73-110.

Campbell-Kelly, M. (2001). Not only microsoft: The maturing of the personal computer software industry: 1982-1995. *Business History Review, 75*(1), 103-146.

Card, D. N., & Glass, R. L. (1990). *Measuring software design quality*. Englewood Cliffs, NJ: Prentice Hall.

Cardenas, A. F. (1977). Technology for automatic generation of application programs. *MIS Quarterly, 1*(3), 49-72.

Carlson, B., Burgess, A., & Miller, C. (1996). *Timeline of computing history*. Retrieved October 21, 2006, from http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/about/history/timeline.pdf

Cavano, J. P., & McCall, J. A. (1978). A framework for the measurement of software quality. *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, San Diego, California, USA,* 133-139.

Cave, W. C., & Salisbury, A. B. (1978). Controlling the software life cycle: The project management task. *IEEE Transactions on Software Engineering, 4*(4), 326-337.

Chen, Q., & Wells, W. D. (1999). Attitude toward the site. *Journal of Advertising Research, 39*(5), 27-37.

Chen, Y., Dios, R., Mili, A., Wu, L., & Wang, K. (2005). An empirical study of programming language trends. *IEEE Software, 22*(3), 72-78.

Cho, N., & Park, S. (2001). Development of electronic commerce user consumer satisfaction index (ECUSI) for internet shopping. *Industrial Management and Data Systems, 101*(8/9), 400-405.

Chung, S. H., Rainer, R. K., & Lewis, B. R. (2003). The impact of information technology infrastructure flexibility on strategic alignment and application implementation. *Communications of AIS, 2003*(11), 191-206.

Clarke, E. M., & Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys*, *28*(4), 626-643.

Coad, P., Lefebvre, E., & De Luca, J. (1999). *Java modeling color with uml: Enterprise components and process*. Upper Saddle River, NJ: Prentice Hall.

Cockburn, A. (2002a). *Agile software development*. Boston, MA: Addison Wesley.

Cockburn, A. (2002b). Learning from agile software development: Part one. *Crosstalk, 15*(10), 10-14.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377-387.

Conte, S. D., Dunsmore, H. E., & Shen, V. Y. (1986). *Software engineering metrics and models*. Menlo Park, CA: Benjamin Cummings.

Coutinho, J. S. (1973). Software reliability growth. *Proceedings of the IEEE Symposium on Computer Software Reliability, New York, New York, USA*, 58-64.

Crosby, P. B. (1979). *Quality is free*. New York, NY: McGraw-Hill.

Cusumano, M. A. (1991). *Japan's software factories: A challenge to U.S. management*. New York, NY: Oxford University Press.

Cusumano, M. A., & Selby, R. W. (1995). *Microsoft secrets: How the world's most powerful software company creates technology, shapes markets, and manages people*. New York, NY: The Free Press.

Cusumano, M. A., & Yoffie, D. B. (1998). *Competing on internet time: Lessons from netscape and its battle with microsoft*. New York, NY: The Free Press.

D'Souza, D. F., & Wills, A. C. (1998). *Objects, components, and frameworks with UML: The catalysis approach*. Reading, MA: Addison Wesley.

Davis, L., Dehning, B., & Stratopoulos, T. (2003). Does the market recognize IT-enabled competitive advantage? *Information and Management, 40*(7), 705-716.

Day, F. W. (1983). Computer aided software engineering (CASE). *Proceedings of the 20th Conference on Design Automation, Miami Beach, Florida, USA*, 129-136.

Denning, J. (1971). Third generation computer systems. *ACM Computing Surveys, 3*(4), 175-216.

Di Benedetto, C. A. (1999). Identifying the key success factors in new product launch. *Journal of Product Innovation Management, 16*(6), 530-544.

Diaz, M., & Sligo, J. (1997). How software process improvement helped motorola. *IEEE Software, 14*(5), 75-81.

Digital Focus. (2006). *Agile 2006 survey: Results and analysis*. Herndon, VA: Author.

Dijkstra, E. W. (1969). *Notes on structured programming* (T.H.-Report 70-WSK-03). Eindhoven, Netherlands: Technological University of Eindhoven.

Dodd, G. G. (1969). Elements of data management systems. *ACM Computing Surveys, 1*(2), 117-133.

Dolk, D. R., & Kirsch, R. A. (1987). A relational information resource dictionary system. *Communications of the ACM, 30*(1), 48-61.

Doll, W. J., & Torkzadeh, G. (1988). The measurement of end user computing satisfaction. *MIS Quarterly, 12*(2), 258-274.

Dreze, X., & Zufryden, F. (1997). Testing web site design and promotional content. *Journal of Advertising Research, 37*(2), 77-91.

DSDM Consortium. (1995). *Dynamic systems development methodology* (Version 2.0). Kent, UK: Dynamic Systems Development Method Consortium.

DSDM. (2007). *DSDM public version 4.2*. Retrieved March 5, 2007, from http://www.dsdm.org

Dunn, O. E. (1966). Information technology: A management problem. *Proceedings of the Third ACM IEEE Conference on Design Automation, New York, NY, USA*, 5.1-5.29.

Dunsmore, H. E., & Gannon, J. D. (1979). Data referencing: An empirical investigation. *IEEE Computer, 12*(12), 50-59.

Dunsmore, H. E., & Gannon, J. D. (1980). Analysis of the effects of programming factors on programming effort. *Journal of Systems and Software, 1*(2), 141-153.

Dzida, W., Herda, S., & Itzfeldt, W. D. (1978). User perceived quality of interactive systems. *Proceedings of the Third International Conference on Software Engineering, Atlanta, Georgia, USA*, 188-195.

Edwards, H. K., & Sridhar, V. (2005). Analysis of software requirements engineering exercises in a global virtual team setup. *Journal of Global Information Management, 13*(2), 21-41.

Ellis, G. H. (1967). The fed's paperless payments mechanism. *Banking, 67*(60), 100-100.

Elshoff, J. L. (1976). An analysis of some commercial PL/1 programs. *IEEE Transactions on Software Engineering, 2*(2), 113-120.

Extreme Programming. (2006). *Extreme programming: A gentle introduction*. Retrieved March 5, 2007, from http://www.extremeprogramming.org

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15*(3), 182-211.

Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development*. London, England: Pearson Education.

Ferguson, P., Humphrey, W. S., Khajenoori, S., Macke, S., & Matvya, A. (1997). Results of applying the personal software process. *IEEE Computer, 30*(5), 24-31.

Fichman, R. G., & Moses, S. A. (1999). An incremental process for software implementation. *Sloan Management Review, 40*(2), 39-52.

Fink, M. (2003). *The business and economics of linux and open source*. Upper Saddle River, NJ: Prentice Hall.

Fisher, A. C. (1968). Computer construction of project networks. *Communications of the ACM, 11*(7), 493-497.

Fitch, A. E. (1969). A user looks at DA: Yesterday, today, and tomorrow. *Proceedings of the Sixth ACM IEEE Conference on Design Automation, New York, NY, USA*, 371-382.

Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering, 31*(7), 529-536.

Fujii, M. S. (1978). A comparison of software assurance methods. *Proceedings of the First Annual Software Quality Assurance Workshop on Functional and Performance Issues, New York, New York, USA*, 27-32.

Fulghum, D. A., & Wall, R. (2004). Is this war? *Aviation Week and Space Technology, 160*(6), 22-24.

Gaffney, J. E. (1981). Metrics in software quality assurance. *Proceedings of the ACM SIGMETRICS Workshop/Symposium on Measurement and Evaluation of Software Quality, Las Vegas, Nevada, USA*, 126-130.

Gane, C. (1987). *Rapid systems development*. New York, NY: Rapid Systems Development, Inc.

Germain, E., & Robillard, P. N. (2005). Engineering based processes and agile methodologies for software development: A comparative case study. *Journal of Systems and Software, 75*(1/2), 17-27.

Gilb, T. (1977). *Software metrics*. Cambridge, MA: Winthrop Publishers.

Goel, A. L., & Okumoto, K. (1979). Time dependent error detection rate model for software and other performance measures. *IEEE Transactions on Reliability, 28*(3), 206-211.

Golden, W., & Powell, P. (2004). Inter organizational information systems as enablers of organizational flexibility. *Technology Analysis and Strategic Management, 16*(3), 299-325.

Goldman, S., Nagel, R., & Preiss, K. (1995). *Agile competitors and virtual organizations: Strategies for enriching the customer*. New York, NY: Van Nostrand Rienhold.

Goodenough, J. B., & Gerhart, S. L. (1975). Toward a theory of test data selection. *Proceedings of the First International Conference on Reliable Software, Los Angeles, California, USA*, 493-510.

Gosain, S., Malhotra, A., & El Sawy, O. A. (2004). Coordinating for flexibility in e-business supply chains. *Journal of Management Information Systems, 21*(3), 7-45.

Goth, G. (2000). The team software process: A quiet quality revolution. *IEEE Software, 17*(6), 125-127.

Gounaris, S., Dimitriadis, S., & Stathakopoulos, V. (2005). Antecedents of perceived quality in the context of internet retail stores. *Journal of Marketing Management, 21*(7/8), 669-700.

Grady, R. B. (1997). *Successful software process improvement*. Upper Saddle River, NJ: Prentice Hall.

Grady, R. B., & Caswell, R. B. (1987). *Software metrics: Establishing a company wide program*. Englewood Cliffs, NJ: Prentice Hall.

Guide International, Inc. (1986). *Joint application design*. Chicago, IL: Author.

Guinan, P. J., Cooprider, J. G., & Faraj, S. (1998). Enabling software development team performance during requirements definition: A behavioral versus technical approach. *Information Systems Research, 9*(2), 101-125.

Guzzo, R. A., & Dickson, M. W. (1996). Teams in organizations: Recent research on performance and effectiveness. *Annual Review of Psychology*, *47*(1), 307-338.

Halloran, T. J., & Scherlis, W. L. (2002). High quality and open source software practices. *Proceedings of the Second Workshop on Open Source Software Engineering, Orlando, Florida, USA*.

Halstead, M. H. (1972). Natural laws controlling algorithm structure? *ACM SIGPLAN Notices, 7*(2), 19-26.

Halstead, M. H. (1977). *Elements of software science*. New York, NY: Elsevier North Holland.

Hansen, W. J. (1978). Measurement of program complexity by the pair (cyclomatic number, operator count). *ACM SIGPLAN Notices, 13*(3), 29-33.

Hardin, K. (1960). Computer automation, work environment, and employee satisfaction: A case study. *Industrial and Labor Relations Review, 13*(4), 559-567.

Hartwick, J., & Barki, H. (1994). Explaining the role of user participation in information system use. *Management Science, 40*(4), 440-465.

Henry, S., & Kafura, D. (1981), Software structure metrics based on information flow. *IEEE Transactions on Software Engineering, 7*(5), 510-518.

Herbold, R. J. (2002). Inside microsoft: Balancing creativity and discipline. *Harvard Business Review, 80*(1), 73-79.

Herbst, P. G. (1962). *Autonomous group functioning*. London, England: Tavistock.

Highsmith, J. A. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. New York, NY: Dorset House.

Highsmith, J. A. (2002). *Agile software development ecosystems*. Boston, MA: Addison Wesley.

Hirschheim, R., Klein, H. K., & Lyytinen, K. (1996). Exploring the intellectual structures of information systems development: A social action theoretic analysis. *Accounting, Management, and Information Technology, 6*(1/2), 1-64.

Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Communications of the ACM, 12*(10), 576-583.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM, 21*(8), 666-677.

Hoegl, M., & Gemuenden, H. G. (2001). Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization Science, 12*(4), 435-449.

Hoisington, S. (1998). Information and analysis at IBM's AS/400 division. *Proceedings of the 10th Annual Minnesota Quality Conference, St. Paul, Minnesota, USA.*

Holibaugh, R., Cohen, S., Kang, K., & Peterson, S. (1989). Reuse: Where to begin and why. *Proceedings of the Conference on Tri-Ada, Pittsburgh, Pennsylvania, USA*, 266-277.

Hong, P., Vonderembse, M. A., Doll, W. J., & Nahm, A. Y. (2005). Role change of design engineers in product development. *Journal of Operations Management, 24*(1), 63-79.

Horowitz, B. B. (1991). *The importance of architecture in DoD software* (Technical Report M91-35). Bedford, MA: The Mitre Corporation.

Huang, X. Z. (1984). The hypergeometric distribution model for predicting the reliability of software. *Microelectronics and Reliability, 24*(1), 11-20.

Humphrey, W. S. (1987). *Characterizing the software process: A maturity framewor*k (CMU/SEI-87-TR-011). Pittsburgh, PA: Software Engineering Institute.

Hunton, J. E., & Beeler, J. O. (1997). Effects of user participation in systems development: A longitudinal field experiment. *MIS Quarterly, 21*(4), 359-388.

Iansiti, M., & MacCormack, A. (1997). Developing products on internet time. *Harvard Business Review, 75*(5), 108-117.

Institute of Electrical and Electronics Engineers. (1990). *IEEE standard glossary of software engineering terminology* (IEEE Std 610.12-1990). New York, NY: Author.

Ives, B., & Olson, M. H. (1984). User involvement and MIS success: A review of research. *Management Science, 30*(5), 586-603.

Ives, B., Olson, M. H., & Baroudi, J. J. (1983). The measurement of user information satisfaction. *Communications of the ACM, 26*(10), 785-793.

Jameson, K. W. (1989). A model for the reuse of software design information. *Proceedings of the 11th International Conference on Software Engineering, Pittsburgh, Pennsylvania, USA*, 205-216.

Janda, S., Trocchia, P. J., & Gwinner, K. P. (2002). Consumer perceptions of internet retail service quality. *International Journal of Service Industry Management, 13*(5), 412-433.

Jelinski, Z., & Moranda, P. B. (1972). Software reliability research. In W. Freiberger (Ed.), *Statistical computer performance evaluation* (pp. 465-484). New York, NY: Academic Press.

Jiang, J. J., Chen, E., & Klein, G. (2002). The importance of building a foundation for user involvement in information system projects. *Project Management Journal, 33*(1), 20-26.

Jin, Z. (2000). How product newness influences learning and probing and the linearity of its development process. *Creativity and Innovation Management, 9*(1), 21-45.

Johnson, L. (1998). A view from the 1960s: How the software industry began. *IEEE Annals of the History of Computing, 20*(1), 36-42.

Johnson, M. (2003). *Agile methodologies: Survey results*. Victoria, Australia: Shine Technologies.

Jones, C. L. (1985). A process-integrated approach to defect prevention. *IBM Systems Journal, 24*(2), 150-165.

Jones, T. C. (1978). Measuring programming quality and productivity. *IBM Systems Journal, 17*(1), 39-63.

Jorgensen, N. (2001). Putting it all in the trunk: Incremental software development in the freebsd open source project. *Information Systems Journal, 11*(4), 321-336.

Joshi, A. W., & Sharma, S. (2004). Customer knowledge development: Antecedents and impact on new product performance. *Journal of Marketing, 68*(4), 47-59.

Joshi, K., Perkins, W. C., & Bostrom, R. P. (1986). Some new factors influencing user information satisfaction: Implications for systems professionals. *Proceedings of the 22nd Annual Computer Personnel Research Conference, Calgary, Canada*, 27-42.

Kalakota, R., & Whinston, A. (1996). *Electronic commerce: A manager's guide*. Reading, MA: Addison Wesley.

Kan, S. H. (1995). *Metrics and models in software quality engineering*. Reading, MA: Addison-Wesley.

Kan, S. H., Dull, S. D., Amundson, D. N., Lindner, R. J., & Hedger, R. J. (1994). AS/400 software quality management. *IBM Systems Journal, 33*(1), 62-88.

Kaufman, S. (1966). The IBM information retrieval center (ITIRC): System techniques and applications. *Proceedings of the 21st National Conference for the Association for Computing Machinery, New York, New York, USA*, 505-512.

Kekre, S., Krishnan, M. S., & Srinivasan, K. (1995). Drivers of customer satisfaction in software products: Implications for design and service support. *Management Science, 41*(9), 1456-1470.

Khermouch, G., & Voight, J. (1997). Sequent computer seeks shop for image work: Uses extreme sports to introduce product. *Adweek Western Edition, 47*(7), 5-5.

Knuth, D. E. (1963). Computer-drawn flowcharts. *Communications of the ACM, 6*(9), 555-563.

Kruchten, P. (2000). *The rational unified process: An introduction*. Reading, MA: Addison Wesley.

Kruchten, P., Obbink, H., & Stafford, J. (2006). The past, present, and future of software architecture. *IEEE Software, 23*(2), 22-30.

Kujala, S. (2003). User involvement: A review of the benefits and challenges. *Behaviour and Information Technology, 22*(1), 1-16.

Langfred, C. W. (2000). The paradox of self-management: Individual and group autonomy in work groups. *Journal of Organizational Behavior, 21*(5), 563-585.

Larman, C. (2004). *Agile and iterative development: A manager's guide*. Boston, MA: Pearson Education.

Leblang, D. B., & Chase, R. P. (1984). Computer aided software engineering in a distributed environment. *Proceedings of the First ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, Pennsylvania, USA*, 104-112.

Leiner, B., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., et al. (1997). The past and future history of the internet. *Communications of the ACM, 40*(2), 102-108.

Li, T., & Calantone, R. J. (1998). The impact of market knowledge competence on new product advantage: Conceptualization and empirical examination. *Journal of Marketing, 62*(4), 13-29.

Lim, J. S., Sharkey, T. W., & Heinrichs, J. H. (2003). New product development practices and export involvement: An initial inquiry. *International Journal of Innovation Management, 7*(4), 475-499.

Lim, W. C. (1998). *Managing software reuse: A comprehensive guide to strategically reengineering the organization for reusable components*. Upper Saddle River, NJ: Prentice Hall.

Lindroos, K. (1997). Use quality and the world wide web. *Information and Software Technology, 39*(12), 827-836.

Lipow, M. (1982). Number of faults per line of code. *IEEE Transactions on Software Engineering, 8*(4), 437-439.

Liskov, B., & Zilles, S. (1974). Programming with abstract data types. *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages, Santa Monica, California, USA*, 50-59.

Littlewood, B. (1979). Software reliability model for modular program structure. *IEEE Transactions on Reliability, 28*(3), 241-246.

Lo, B. (1992). *Syntactical construct based APAR projection* (Technical Report). San Jose, CA: IBM Santa Teresa Research Laboratory.

Lombardi, L. (1961). Theory of files. *Communications of the ACM, 4*(7), 324-324.

Lucas, H. C. (1973). User reactions and the management of information services. *Management Informatics, 2*(4), 165-162.

Lucas, H. C. (1974). Measuring employee reactions to computer operations. *Sloan Management Review, 15*(3), 59-67.

Lundquist, E. (1996). When is internet time just too fast? *PC Week, 13*(32), 102-102.

Lynch, J. E. (1990). The impact of electronic point of sale technology (EPOS) on marketing strategy and retailer-supplier relationships. *Journal of Marketing Management, 6*(2), 157-168.

Lynn, G. S., Skov, R. B., & Abel, K. D. (1999). Practices that support team learning and their impact on speed to market and new product success. *Journal of Product Innovation Management, 16*(5), 439-454.

Lyons, M. L. (1980). Measuring user satisfaction: The semantic differential technique. *Proceedings of the 17th Annual Conference on Computer Personnel Research, Miami, Florida, USA*, 79-87.

MacCormack, A. (1998). *Managing adaptation: An empirical study of product development in rapidly changing environments*. Unpublished doctoral dissertation, Harvard University, Boston, MA, United States.

MacCormack, A. (2001). Product development practices that work: How internet companies build software. *MIT Sloan Management Review, 42*(2), 15-24.

MacCormack, A., Verganti, R., & Iansiti, M. (2001). Developing products on internet time: The anatomy of a flexible development process. *Management Science, 47*(1), 133-150.

Madden, W., & Rone, K. (1984). Design, development, integration: Space shuttle flight software system. *Communications of the ACM, 27*(9), 914-925.

Maish, A. M. (1979). A user's behavior toward his MIS. *MIS Quarterly, 3*(1), 39-52.

Mandell, L. (1977). Diffusion of EFTS among national banks. *Journal of Money, Credit, and Banking, 9*(2), 341-348.

Martin, J. (1965). *Programming real-time computer systems*. Englewood Cliffs, NJ: Prentice Hall.

Martin, J. (1991). *Rapid application development*. New York, NY: Macmillan.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308-320.

McIlroy, M. D. (1960). Macro instruction extensions of compiler languages. *Communications of the ACM, 3*(4), 214-220.

McIntyre, S. C., & Higgins, L. F. (1988). Object oriented systems analysis and design: Methodology and application. *Journal of Management Information Systems, 5*(1), 25-35.

McKeen, J. D., Guimaraes, T., & Wetherbe, J. C. (1994). The relationship between user participation and user satisfaction: An investigation of four contingency factors. *MIS Quarterly, 18*(4), 427-451.

McKinney, V., Yoon, K., & Zahedi, F. (2002). The measurement of web customer satisfaction: An expectation and disconfirmation approach. *Information Systems Research, 13*(3), 296-315.

Melnik, G., & Maurer, F. (2005). A cross program investigation of student's perceptions of agile methods. *Proceedings of the 27th International Conference on Software Engineering, St. Louis, Missouri, USA*, 481-488.

Merwin, R. E. (1972). Estimating software development schedules and costs. *Proceedings of the Ninth Annual ACM IEEE Conference on Design Automation, New York, NY, USA*, 1-4.

Middleton, R., & Wardley, P. (1990). Information technology in economic and social history: The computer as philosopher's stone or pandora's box? *Economic History Review, 43*(4), 667-696.

Miller, L. A. (1974). Programming by non-programmers. *International Journal of Man-Machine Studies, 6*(2), 237-260.

Millington, D., & Stapleton, J. (1995). Developing a RAD standard. *IEEE Software, 12*(5), 54-56.

Mills, H. D. (1971). *Chief programmer teams: Principles and procedures* (IBM Rep. FSC 71-5108). Gaithersburg, MD: IBM Federal Systems Division.

Mills, H. D., Linger, R. C., & Hevner, A. R. (1987). Box structured information systems. *IBM Systems Journal, 26*(4), 395-413.

Milne, M.A. (1971). CLUSTR: A program for structuring design problems. *Proceedings of the Eighth Annual ACM IEEE Design Automation Conference, Atlantic City, New Jersey, USA*, 242-249.

Montalbano, M. (1962). Tables, flowcharts, and program logic. *IBM Systems Journal, 1*(1), 51-63.

Moranda, P. B. (1979). An error detection model for application during software development. *IEEE Transactions on Reliability, 28*(5), 325–329.

Motley, R. W., & Brooks, W. D. (1977). *Statistical prediction of programming errors* (RADC-TR-77-175). Griffis AFB, NY: Rome Air Development Center.

Mowery, D. C., & Simcoe, T. (2002). Is the internet a US invention? An economic and technological history of computer networking. *Research Policy, 31*(8/9), 1369-1387.

Musa, J. D. (1999). *Software reliability engineering*. New York, NY: McGraw Hill.

Musa, J. D., Iannino, A., & Okumoto, K. (1987). *Software reliability: Measurement, prediction, and application*. New York, NY: McGraw Hill.

Myers, G. J. (1976). *Software reliability: Principles and practices*. New York, NY: John Wiley & Sons.

Myers, G. J. (1977). An extension to the cyclomatic measure of program complexity. *SIGPLAN Notices, 12*(10), 61-64.

Naumann, J. D., & Jenkins, A. M. (1982). Prototyping: The new paradigm for systems development. *MIS Quarterly, 6*(3), 29-44.

Neighbors, J. M. (1984). The draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering, 10*(5), 564-574.

Nelson, K. M., & Cooprider, J. G. (2001). The relationship of software system flexibility to software system and team performance. *Proceedings of the 22nd International Conference on Information Systems (ICIS 2001), New Orleans, Louisiana, USA,* 23-32.

Nerlove, M. (2004). Programming languages: A short history for economists. *Journal of Economic and Social Measurement, 29*(1-3), 189-203.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 73-78.

New York Stock Exchange (2006). *NYSE timeline of technology*. Retrieved October 23, 2006, from http://www.nyse.com/about/history/timeline_technology.html

Northrop, L. M. (2002). SEI's software product line tenets. *IEEE Software, 19*(4), 32-40.

Notkin, D. (1989). The relationship between software development environments and the software process. *Proceedings of the Third ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Massachusetts, USA*, 107-109.

O'Connor, C. M. (2005). Sarbanes oxley: Frying the small fry. *Investment Dealers Digest, 71*(25), 10-11.

O'Reilly, T. (1999). Lessons from open source software development. *Communications of the ACM, 42*(4), 32-37.

O'Reilly, T. (2006). *What is web 2.0: Design patterns and business models for the next generation of software*. Retrieved December 10, 2006, from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Ohba, M. (1984). Software reliability analysis models. *IBM Journal of Research and Development, 21*(4), 428-443.

Oldfather, P. M., Ginsberg, A. S., & Markowitz, H. M. (1966). *Programming by questionnaire: How to construct a program generator* (RM-5128-PR). Santa Monica, CA: The RAND Corporation.

Oviedo, E. I. (1980). Control flow, data flow, and program complexity. *Proceedings of the Fourth International IEEE Computer Software and Applications Conference (COMPSAC 1980), Chicago, Illinois, USA*, 146-152.

Palanisamy, R., & Sushil. (2003). Measurement and enablement of information systems for organizational flexibility: An empirical study. *Journal of Services Research, 3*(2), 81-103.

Palmer, S. R., & Felsing, J. M. (2002). *A practical guide to feature driven development*. Upper Saddle River, NJ: Prentice Hall.

Panzl, D. J. (1976). Test procedures: A new approach to software verification. *Proceedings of the Second International Conference on Reliable Software, San Francisco, California, USA*, 477-485.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM, 15*(12), 1053-1058.

Paul, S., Samarah, I. M., Seetharaman, P., & Mykytyn, P. P. (2004). An empirical investigation of collaborative conflict management style in group support system based global virtual teams. *Journal of Management Information Systems, 21*(3), 185-222.

Pavlicek, R. C. (2000). *Embracing insanity: Open source software development*. Indianapolis, IN: Sams Publishing.

Pearson, S. W., & Bailey, J. E. (1980). Measurement of computer user satisfaction. *ACM SIGMETRICS Performance Evaluation Review, 9*(1), 9-68.

Pham, H. (2000). *Software reliability*. Singapore: Springer Verlag.

Pigott, D. (2006). *HOPL: An interactive roster of programming languages*. Retrieved October 21, 2006, from http://hopl.murdoch.edu.au

Pine, B. J. (1989). Design, test, and validation of the application system/400 through early user involvement. *IBM System Journal, 28*(3), 376-385.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit for software development managers*. Boston, MA: Addison Wesley.

Potier, D., Albin, J. L., Ferreol, R., & Bilodeau, A. (1982). Experiments with computer software complexity and reliability. *Proceedings of the Sixth International Conference on Software Engineering, Tokyo, Japan*, 94-103.

Prewitt, E. (2004). The agile 100. *CIO Magazine, 17*(21), 4-7.

Prieto-Diaz, R. (1987). Domain analysis for reusability. *Proceedings of the 11th Annual International Computer Software and Applications Conference (COMPSAC 1987), Tokyo, Japan*, 23-29.

Przasnyski, Z. H., & Tai, L. S. (1999). Stock market reaction to malcolm baldridge national quality award announcements: Does quality pay? *Total Quality Management, 10*(3), 391-400.

Radice, R. A., Harding, J. T., Munnis, P. E., & Phillips, R. W. (1985). A programming process study. *IBM Systems Journal, 24*(2), 91-101.

Reid, R. H. (1997). *Architects of the web: 1,000 days that build the future of business.* New York, NY: John Wiley & Sons.

Reifer, D. J. (2003). The business case for agile methods/extreme programming (XP). *Proceedings of the Seventh Annual PSM Users Group Conference, Keystone, Colorado, USA*, 1-30.

Rigby, P. J., Stoddart, A. G., & Norris, M. T. (1990). Assuring quality in software: Practical experiences in attaining ISO 9001. *British Telecommunications Engineering, 8*(4), 244-249.

Riordan, C. M., & Weatherly, E. W. (1999). Defining and measuring employee's identification with their work groups. *Educational and Psychological Measurement, 59*(2), 310-324.

Rondeau, P. J., Ragu-Nathan, T. S., & Vonderembse, M. A. (2006). How involvement, IS management effectiveness, and end user computing impact IS performance in manufacturing firms. *Information and Management, 43*(1), 93-107.

Rosen, S. (1969). Electronic computers: A historical survey. *ACM Computing Surveys, 1*(1), 7-36.

Rosson, M. B., & Alpert, S. R. (1990). The cognitive consequences of object oriented design. *Human Computer Interaction, 5*(4), 345-379.

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of the Western Electronic Show and Convention (WESCON 1970), Los Angeles, California, USA*, 1-9.

Rubey, R. J., & Hartwick, R. D. (1968). Quantitative measurement of program quality. *Proceedings of the 23rd ACM National Conference, Washington, DC, USA*, 671-677.

Sammet, J. E. (1972a). An overview of programming languages for special application areas. *Proceedings of the Spring Joint American Federation of Information Processing Societies Conference (AFIPS 1972), Montvale, New Jersey, USA*, 299-311.

Sammet, J. E. (1972b). Programming languages: History and future. *Communications of the ACM, 15*(7), 601-610.

Sarin, S., & McDermott, C. (2003). The effect of team leader characteristics on learning, knowledge application, and performance of cross functional new product development teams. *Decision Sciences, 34*(4), 707-739.

Sassenburg, H. (2002). The nonsense of ROI. *SPIder Koerier, 2*(4), 4-5.

Sauer, C., Jeffery, D. R., Land, L., & Yetton, P. (2000). The effectiveness of software development technical reviews: A behaviorally motivated program of research. *IEEE Transactions on Software Engineering, 26*(1), 1-15.

Sawyer, S. (2001). Effects of intra group conflict on packaged software development team performance. *Information Systems Journal, 11*(2), 155-178.

Schick, G. J., & Wolverton, R. W. (1978). An analysis of competing software reliability analysis models. *IEEE Transactions on Software Engineering, 4*(2), 104-120.

Schneidewind, N. F., & Hoffmann, H. (1979). An experiment in software error data collection and analysis. *IEEE Transactions on Software Engineering, 5*(3), 276-286.

Schulze, A., & Hoegl, M. (2006). Knowledge creation in new product development projects. *Journal of Management, 32*(2), 210-236.

Schwaber, K. (1995). Scrum development process. *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 1995), Austin, Texas, USA*, 117-134.

Schwaber, K. (2004). *Agile project management with scrum*. Redmond, WA: Microsoft Press.

Selz, D., & Schubert, P. (1997). Web assessment: A model for the evaluation and the assessment of successful electronic commerce applications. *Electronic Markets, 7*(3), 46-48.

Shapiro, S. (1997). Splitting the difference: The historical necessity of synthesis in software engineering. *IEEE Annals of the History of Computing, 19*(1), 20-54.

Shen, V. Y., Yu, T. J., Thebaut, S. M., & Paulsen, L. R. (1985). Identifying error prone software: An empirical study. *IEEE Transactions on Software Engineering, 11*(4), 317-324.

Sherman, J. D., Souder, W. E., & Jenssen, S. A. (2000). Differential effects of the primary forms of cross functional integration on product development cycle time. *Journal of Product Innovation Management, 17*(4), 257-267.

Shooman, M. L. (1983). *Software engineering*. New York, NY: McGraw Hill.

Shooman, M. L., & Bolsky, M. I. (1975). Types, distribution, and test and correction times for programming errors. *Proceedings of the International Conference on Reliable Software, Los Angeles, California, USA*, 347-357.

Singh, N., & Sushil. (2004). Flexibility in product development for success in dynamic market environment. *Global Journal of Flexible Systems Management, 5*(1), 23-34.

Smith, A. (1988). EDI: Will banks be odd man out? *ABA Banking Journal, 80*(11), 77-79.

Standish Group. (2000). *Extreme chaos*. West Yarmouth, MA: Author.

Steinmueller, W. E. (1996). The U.S. software industry: An analysis and interpretive history. In D. C. Mowery (Ed.), *The international computer software industry* (pp. 25-52). New York, NY: Oxford University Press.

Stevens, W. P., Myers, G. J., & Constantine, L. L. (1974). Structured design. *IBM Systems Journal, 13*(2), 115-139.

Stewart, K. J., & Gosain, S. (2006). The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly, 30*(2), 291-314.

Stump, R. L., Athaide, G. A., & Joshi, A. W. (2002). Managing seller buyer new product development relationships for customized products: A contingency model based on transaction cost analysis and empirical test. *Journal of Product Innovation Management, 19*(6), 439-454.

Sukert, A. N. (1979). Empirical validation of three software error prediction models. *IEEE Transactions on Reliability, 28*(3), 199-205.

Sulack, R. A., Lindner, R. J., & Dietz, D. N. (1989). A new development rhythm for AS/400 software. *IBM Systems Journal, 28*(3), 386-406.

Sunazuka, T., Azuma, M., & Yamagishi, N. (1985). Software quality assessment technology. *Proceedings of the Eighth International Conference on Software Engineering, London, England*, 142-148.

Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Agile Project Management Advisory Service Executive Update, 5*(2), 1-4.

Swanson, E. B. (1976). The dimensions of maintenance. *Proceedings of the Second International Conference on Software Engineering, San Francisco, California, USA*, 492-497.

Szymanski, D. M., & Hise, R. T. (2000). E-satisfaction: An initial examination. *Journal of Retailing, 76*(3), 309-322.

Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review, 64*(1), 137-146.

Tanenbaum, A. (2001). *Modern operating systems*. Englewood Cliffs, NJ: Prentice Hall.

Tang, V., & Collar, E. (1992). IBM AS/400 new product launch process ensures satisfaction. *Long Range Planning*, 25(1), 22-27.

Teichroew, D., & Sayani, H. (1971). Automation of system building. *Datamation, 17*(16), 25-30.

Teorey, T. J., & Fry, J. P. (1980). The logical record access approach to database design. *ACM Computing Surveys, 12*(2), 179-211.

Thayer, C. H. (1958). Automation and the problems of management. *Vital Speeches of the Day, 25*(4), 121-125.

Thomas, D. (2005). Agile programming: Design to accommodate change. *IEEE Software, 22*(3), 14-16.

Thomke, S., & Reinertsen, D. (1998). Agile product development: Managing development flexibility in uncertain environments. *California Management Review, 41*(1), 8-30.

Tiwana, A., & McLean, E. R. (2005). Expertise integration and creativity in information systems development. *Journal of Management Information Systems, 22*(1), 13-43.

Turk, D., France, R., & Rumpe, B. (2005). Assumptions underying agile software development processes. *Journal of Database Management, 16*(4), 62-87.

U.S. Census Bureau. (2006). *E-stats*. Washington, DC: Author.

U.S. Department of Commerce. (2003). *Digital economy*. Washington, DC: Author.

U.S. Department of Commerce. (2006). *Information and communication technology: 2004*. Washington, DC: Author.

U.S. Department of Defense. (1976). *Military standard: Technical reviews and audits for systems, equipments, and computer software* (MIL-STD-1521A). Hanscom AFB, MA: Author.

Van Den Bosch, F., Ellis, J. R., Freeman, P., Johnson, L., McClure, C. L., Robinson, D., et al. (1982). Evaluation of software development life cycle: Methodology implementation. *ACM SIGSOFT Software Engineering Notes, 7*(1), 45-60.

Version One. (2006). *The state of agile development*. Apharetta, GA: Author.

Vise, D. A. (2005). *The google story: Inside the hottest business, media, and technology success of our time*. New York, NY: Delcorte Press.

Waldstein, N. S. (1974). *The walk thru: A method of specification design and review* (TR 00.2536). Poughkeepsie, NY: IBM Corporation.

Wall, J. K., & Ferguson, P. A. (1977). Pragmatic software reliability prediction. *Proceedings of the Annual Reliability and Maintainability Symposium, Piscataway, New Jersey, USA*, 485-488.

Walsh, D. A. (1977). Structured testing. *Datamation, 23*(7), 111-111.

Walsh, M. D. (1982). Evaluating user satisfaction. *Proceedings of the 10th Annual ACM SIGUCCS Conference on User Services, Chicago, Illinois, USA*, 87-95.

Weber, C., Paulk, M., Wise, C., & Withey, J. (1991). *Key practices of the capability maturity model* (CMU/SEI-91-TR-025). Pittsburgh, PA: Software Engineering Institute.

Wegner, P. (1972). The vienna definition language. *ACM Computing Surveys, 4*(1), 5-63.

Wegner, P. (1980). The ada language and environment. *ACM SIGSOFT Software Engineering Notes, 5*(2), 8-14.

Wegner, P., Scherlis, W., Purtilo, J., Luckham, D., & Johnson, R. (1992). Object oriented megaprogramming. *Proceedings on Object Oriented Programming Systems, Languages, and Applications, Vancouver, British Columbia, Canada*, 392-396.

Weinberg, G. M. (1971). *The psychology of computer programming*. New York, NY: Van Nostrand Reinhold.

Weinberg, G. M., & Gressett, G. L. (1963). An experiment in automatic verification of programs. *Communications of the ACM, 6*(10), 610-613.

Weissman, L. (1973). Psychological complexity of computer programs. *ACM SIGPLAN Notices, 8*(6), 92-95.

Whittaker, J. A. (2000). What is software testing? And why is it so hard? *IEEE Software, 17*(1), 70-79.

Williams, L., & Kessler, R. (2003). *Pair programming illuminated*. Boston, MA: Pearson Education.

Wirth, N. (1971). Program development by stepwise refinement. *Communications of the ACM, 14*(4), 221-227.

Wolfinbarger, M., & Gilly, M. C. (2003). Etailq: Dimensionalizing, measuring, and predicting etail quality. *Journal of Retailing, 79*(3), 183-198.

Woodward, M. R., Hennell, M. A., & Hedley, D. (1979). A measure of control flow complexity in program text. *IEEE Transactions on Software Engineering, 5*(1), 45-50.

Xia, W., & Lee, G., (2005). Complexity of information systems development projects: Conceptualization and measurement development. *Journal of Management Information Systems, 22*(1), 45-83.

Yamada, S., Ohba, M., & Osaki, S. (1983). S shaped reliability growth modeling for software error prediction. *IEEE Transactions on Reliability, 32*(5), 475-478.

Yoffie, D. B., & Cusumano, M. A. (1999). Building a company on Internet time: Lessons from netscape. *California Management Review, 41*(3), 8-28.

Yoo, B., & Donthu, N. (2001). Developing a scale to measure the perceived quality of an internet shopping site (sitequal). *Quarterly Journal of Electronic Commerce, 2*(1), 31-45.

Youngs, E. A. (1970). *Error proneness in programming*. Unpublished doctoral dissertation. University of North Carolina at Chapel Hill, Chapel Hill, NC, United States.

Yourdon, E. (1976). The emergence of structured analysis. *Computer Decisions*, *8*(4), 58-59.

Zhang, M. J. (2005). Information systems, strategic flexibility, and firm performance: An empirical investigation. *Journal of Engineering and Technology Management*, *22*(3), 163-184.

Zhao, L., & Deek, F. P. (2004). User collaboration in open source software development. *Electronic Markets, 14*(2), 89-103.

Zolnowski, J. C., & Simmons, D. B. (1981). Taking the measure of program complexity. *Proceedings of the AFIPS National Computer Conference, Chicago, Illinois, USA*, 329-336.

APPENDICES

Appendix A — Agile Methods Survey Instrument

Iterative Development

1. We develop software using time-based iterations, increments, or demonstrations.
2. We develop software using operational iterations, increments, or demonstrations (working code).
3. We develop software using small iterations, increments, or demonstrations.
4. We develop software using daily, weekly, bi-weekly, or monthly iterations, increments, or demonstrations.
5. We develop software using multiple (several) iterations, increments, or demonstrations.

Customer Feedback

6. We seek customer feedback on our software iterations, increments, or demonstrations.
7. We receive customer feedback on our software iterations, increments, or demonstrations.
8. We receive timely customer feedback on our software iterations, increments, or demonstrations.
9. We receive a lot of (detailed) customer feedback on our software iterations, increments, or demonstrations.
10. We incorporate customer feedback into our software iterations, increments, or demonstrations.

Well-Structured Teams

11. Our software teams have clear administrative or technical leaders.
12. Our software teams have clear visions, missions, or strategies.
13. Our software teams have clear goals or objectives.
14. Our software teams have clear schedules or timelines.
15. Our software teams have a small size with no more than 10 people.

Flexibility

16. Our software is designed to be as small as possible.
17. Our software is designed to be as simple as possible.
18. Our software is designed to be modular or object-oriented.
19. Our software is designed to work on multiple operating systems.
20. Our software is designed to be changed, modified, or maintained.

Appendix B — Website Quality Survey Instrument

Website Design (Wolfinbarger & Gilly, 2003)

1. The website provides in-depth information.
2. The site doesn't waste my time.
3. It is quick and easy to complete a transaction at this website.
4. The level of personalization at site is about right, not too much or too little.
5. This website has good selection.

Privacy and Security (Wolfinbarger & Gilly, 2003)

6. I feel like my privacy is protected at this site.
7. I feel safe in my transactions with this website.
8. The website has adequate security features.

Fulfillment and Reliability (Wolfinbarger & Gilly, 2003)

9. You get what you ordered from this site.
10. The product is delivered by the time promised by the company.
11. The product that came was represented accurately by the websitee.

Customer Service (Wolfinbarger & Gilly, 2003)

12. The company is willing and ready to respond to customer needs.
13. When you have a problem, the website shows a sincere interest in solving it.
14. Inquiries are answered promptly.

Appendix C — Research Project Notification and Human Subjects Protection Form

As outlined in UMUC Policy 130.25, Conducting Research Involving Human Subjects, all UMUC students, staff, faculty, and individuals external to UMUC, who wish to conduct research involving human subjects must adhere to this policy before conducting any research.

All researchers must complete the Research Project Notification and Human Subjects Protection Form and receive all appropriate signatures before initiating any research. Please fill in each blank and provide copies of any additional documents (i.e. draft questionnaires) as necessary.

If you have any questions about the form, ask your IRB representative.

Research Project Notification and Human Subjects Protection Form

Date: _____April 4, 2007_____

Name of Proposer:____David F. Rico_____

Email: _dave@davidfrico.com__ Phone:___410-551-3813___

Fax: _____

Unit Representative: _____

Email: _____Phone: _____

Fax: _____

---

1. Title of Project:

__Effects of Agile Methods on Website Quality for Electronic Commerce___

2. Purpose of the Project:

The purpose of this project is to survey information systems managers to measure their adherence to using agile methods to manage the development of Internet websites and subsequent website quality.

3. Survey Instruments or Data Collection Methodology to be used.

A. Interview, focus group, questionnaire, or other?

___Interview, Questionnaire_____
If other, please explain.

B. Online, regular mail, face-to-face, or other ?

__Face-to-face cognitive interviews to validate instrument, Online survey for data collection_

If other, please explain.

C. Please attach a copy of each instrument if it has been prepared. If not, explain how it will be developed.

See Appendix A - Agile Methods Survey Instrument and Appendix B - Website Quality Survey

4. Research Design including population and sample, if applicable.

A. UMUC Students, Staff, Faculty, or other?

__Software Developers, Graduate Students__
If other, please explain.

B. Sample Size_6 for cognitive interviews, 1,800 for agility, 200 for website quality survey_

C. What information will be collected (e.g. any potentially sensitive subjects such as drug use, etc) ?

a. Stage 0: Reactions of 6 programmers to survey instruments.
b. Stage I: Software development practices of 1,800 programmers.
c. Stage II: Assessments of website quality by 200 graduate students.

D. What data other than the respondents' answers will be sought (e.g. via access to UMUC records)?

n/a

E. How will the privacy and confidentiality of the human subjects be recorded and stored?

a. No personal identity data will be recorded (e.g., firm, address, phone numbers, addresses, etc.).
b. Responses to surveys will be stored on researchers personal computer for statistical analysis.
c. URLs of websites (e.g., http://www.google.com) will not be revealed.

F. How will the analysis protect the respondent's right to privacy (e.g., if less than five people are in a table cell, how will their identity remain anonymous)?

a. No personal identity data will be recorded (e.g., firm, address, phone numbers, addresses, etc.).
b. Responses to surveys will be stored on researchers personal computer for statistical analysis.
c. URLs of websites (e.g., http://www.google.com) will not be revealed.

5. Signature Approvals

Proposer Verification

__X__I have read and understand the UMUC Policy 130.25 and the procedures for completing the Research Project Notification and Human Subjects Protection Form. I have completed the Research Project Notification and Human Subjects Protection Form fully and accurately. I agree to comply with UMUC Policy 130.25.

_____ _____
Research Proposer Date

Course Instructor Approval

_____I have determined that the proposed research involves minimal risk and is limited to individuals within the proposer's course and is approved.

___X___I have determined that there may be a risk to human subjects and will forward the application form to the appropriate IRB representative.

_____ _____
Course Instructor Date

Please return form to the Proposer or forward to IRB representative, as appropriate.

IRB Representative Approval

_____I have determined that the proposed research involves minimal risk and is approved.

_____I have determined that the proposed research involves more than minimal risk and/or is being completed by an individual external to UMUC and must be forwarded to the IRB to complete a full review.

_____ _____
IRB Representative Date

Please return form to the Proposer within 7 working days or forward to full IRB, as appropriate.

IRB Full Board Approval

The IRB has completed a full review. The proposed research has been:

_____ approved. _____ disapproved.

Explanation:

_____ _____
IRB Chairperson Date

Please return form to the Proposer within 10 working days.

Appropriate Research Office Approval

_____I have determined that the proposed research will not denigrate the integrity of UMUC data collection nor will it put an undue burden on UMUC and UMUC data collection and surveying.

_____I have determined that the proposed research should not be undertaken. Please see the attached memo for further explanation of this decision.

Please indicate with an "X" which individual has made the determination and then sign on the line provided below:
_____ Vice President, Accountability and Planning, UMUC-Adelphi
_____ Director, Office of Institutional Research, UMUC-Asia
_____ Director, Office of Institutional Planning, UMUC-Europe

_____ _____
Appropriate Research Staff Date

Please return form to the Proposer within 7 working days.

Abstracts of all proposals will be kept for two years in the UMUC-Adelphi Office of Institutional Accountability, Planning, and Research. All completed research forms, electronic research files, proposals, research reports and data collection instruments must be stored in the UMUC-Adelphi Office of Institutional Accountability, Planning, and Research for two years.