

A COMPARISON AND ANALYSIS OF AGILE METHODS

The purpose of this paper 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 1 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.

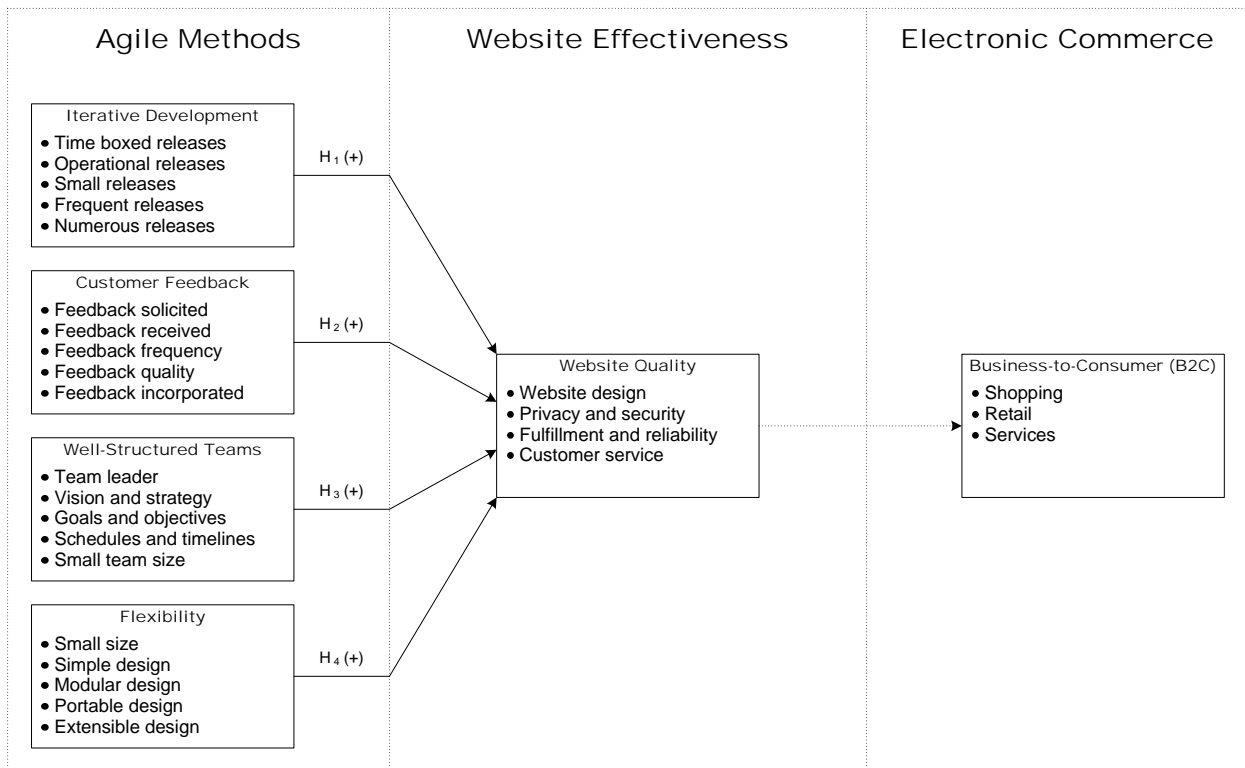


Figure 1. Conceptual model of agile methods, website quality, and e-commerce.

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, 2002). “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 (Wolfenbarger & 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 1. 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 2. 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, Lefebvre, & De Luca, 1999). Originally, FDD only had three roles: (a) chief programmer, (b) class owner, and (c) feature team (Coad, Lefebvre, & 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 3. 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 3. 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 3. 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 4. 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 5. 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 6. 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). Synchron-stabilize 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 7 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 7 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 7. 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) scalable, 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 (H₁): 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 (H₂): 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, Coopriider, & 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 (H₃): 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 & Coopriider, 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 (H₄): 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.

REFERENCES

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis* (478). Oulu, Finland: VTT Publications.
- Agile Manifesto. (2001). *Manifesto for agile software development*. Retrieved November 29, 2006, from <http://www.agilemanifesto.org>
- Allen, T. J. (1966). Studies of the problem solving process in engineering design. *IEEE Transactions on Engineering Management*, 13(2), 72-83.
- 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.
- Arambewela, R., & Hall, J. (2006). A comparative analysis of international education satisfaction using servqual [Special issue]. *Journal of Services Research*, 6, 141-163.
- 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.
- 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.
- Bauer, R. A., Collar, E., & Tang, V. (1992). *The silverlake project: Transformation at IBM*. New York, NY: Oxford University Press.
- 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.
- Bretthauer, D. (2002). Open source software: A history. *Information Technology and Libraries*, 21(1), 3-10.

- 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.
- Callahan, R. E. (1979). A management dilemma revisited: Must businesses choose between stability and adaptability? *Sloan Management Review*, 21(1), 25-33.
- 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.
- 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. (2002). Learning from agile software development: Part one. *Crosstalk*, 15(10), 10-14.
- 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.
- Di Benedetto, C. A. (1999). Identifying the key success factors in new product launch. *Journal of Product Innovation Management*, 16(6), 530-544.
- 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>
- 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.
- Extreme Programming. (2006). *Extreme programming: A gentle introduction*. Retrieved March 5, 2007, from <http://www.extremeprogramming.org>
- Feller, J., & Fitzgerald, B. (2002). *Understanding open source software development*. London, England: Pearson Education
- Fink, M. (2003). *The business and economics of linux and open source*. Upper Saddle River, NJ: Prentice Hall.

- 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.
- 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.
- 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.
- 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.
- Guinan, P. J., Coopridge, 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*.
- Hartwick, J., & Barki, H. (1994). Explaining the role of user participation in information system use. *Management Science*, 40(4), 440-465.
- 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. (2002). *Agile software development ecosystems*. Boston, MA: Addison Wesley.
- 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*.
- 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.

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

- 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.
- 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.
- 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.
- 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.
- Millington, D., & Stapleton, J. (1995). Developing a rad standard. *IEEE Software*, 12(5), 54-56.
- Nelson, K. M., & Coopridge, 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.
- Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 73-78.
- 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.
- 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.
- Pine, B. J. (1989). Design, test, and validation of the application system/400 through early user involvement. *IBM System Journal*, 28(3), 376-385.

- 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.
- 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.
- Sawyer, S. (2001). Effects of intra group conflict on packaged software development team performance. *Information Systems Journal*, 11(2), 155-178.
- Schulze, A., & Hoegl, M. (2006). Knowledge creation in new product development projects. *Journal of Management*, 32(2), 210-236.
- Schwaber, K. (2004). *Agile project management with scrum*. Redmond, WA: Microsoft Press.
- Sherman, J. D., Souder, W. E., & Janssen, 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.
- Singh, N., & Sushil. (2004). Flexibility in product development for success in dynamic market environment. *Global Journal of Flexible Systems Management*, 5(1), 23-34.
- 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.
- 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.
- Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Agile Project Management Advisory Service Executive Update*, 5(2), 1-4.
- Takeuchi, H., & Nonaka, I. (1986). The new product development game. *Harvard Business Review*, 64(1), 137-146.
- Tang, V., & Collar, E. (1992). IBM AS/400 new product launch process ensures satisfaction. *Long Range Planning*, 25(1), 22-27.
- 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 underlying agile software development processes. *Journal of Database Management*, 16(4), 62-87.
- Williams, L., & Kessler, R. (2003). *Pair programming illuminated*. Boston, MA: Pearson Education.
- Wolfenbarger, M., & Gilly, M. C. (2003). Etailq: Dimensionalizing, measuring, and predictingetail quality. *Journal of Retailing*, 79(3), 183-198.
- Xia, W., & Lee, G., (2005). Complexity of information systems development projects: Conceptualization and measurement development. *Journal of Management Information Systems*, 22(1), 45-83.
- Yoffie, D. B., & Cusumano, M. A. (1999). Building a company on Internet time: Lessons from netscape. *California Management Review*, 41(3), 8-28.
- 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.