

USE OF AGILE METHODS IN SOFTWARE ENGINEERING EDUCATION

by Dr. David F. Rico and Dr. Hasan H. Sayani

Abstract

Although agile methods have been occasionally used in higher education for the last six years, the use of agile methods in software engineering education is still a relatively new phenomenon. The purpose of this article is to describe the introduction of agile methods in a capstone course for a masters degree in software engineering at the University of Maryland University College. Three virtual distributed teams of five students were tasked with using agile methods to develop competing electronic commerce websites over a period of 13 weeks in the Fall semester of 2008. With little training in agile methods, virtual teams, web design, or distributed tools, the students used their formal problem solving skills to successfully complete three fully functional websites. While the students were particularly adept at addressing the technical aspects of agile methods, more emphasis should be placed on softer issues such as teamwork and customer collaboration.

Introduction

Since the emergence of the software engineering discipline in 1968, the U.S. software industry has grown in revenues from \$70 million into the \$330 billion blockbuster industry it is today.⁴ Born out of crisis, today software engineering is defined as the “the application of a systematic, disciplined, and quantifiable approach to development, operation, and maintenance of software.” The software crisis was the result of the commercialization of mainframe computers and the sudden demand for software needing years, billions of dollars, and thousands of people to create. Likewise, the software crisis resulted in the impetus to provide formal education and academic degrees in computer science in the 1960s and the first software engineering degrees in the 1970s. The software industry entered another growth period with the rise of the Internet, as the number of websites, users, and revenues grew to 136 million, 1.3 billion, and \$220 billion, respectively.⁴⁰ Information technology is the second leading contributor to the U.S. economy, as well as the top 10 industrialized nations, and software engineering became the third fastest growing profession. Numerous online software engineering programs appeared and existing universities adapted their computer science and software engineering programs to Web technologies and associated skills.

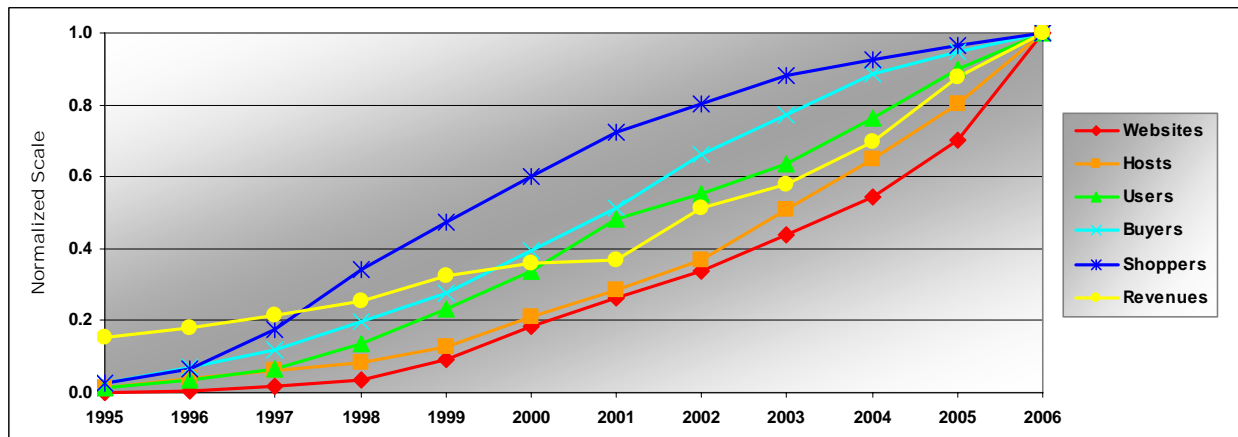


Figure 1. Information technology and Internet industry growth (from 1995 to 2006)

Agile Methods

Agile methods are contemporary software engineering approaches based on teamwork, customer collaboration, iterative development, and people, process, and technology adaptable to change.¹² Traditional methods are software engineering approaches based on highly structured project plans, exhaustive documentation, and extremely rigid processes designed to minimize change. Agile methods are a de-evolution of management thought predating the industrial revolution and use craft industry principles like artisans creating made-to-order items for individual customers. Traditional methods represent the amalgamation of management thought over the last century and use scientific management principles such as efficient production of items for mass markets. Agile methods are new product development processes that have the ability to bring innovative products to market quickly and inexpensively on complex projects with ill-defined requirements. Traditional methods resemble manufacturing processes that have the ability to economically and efficiently produce high quality products on projects with stable and well-defined requirements.

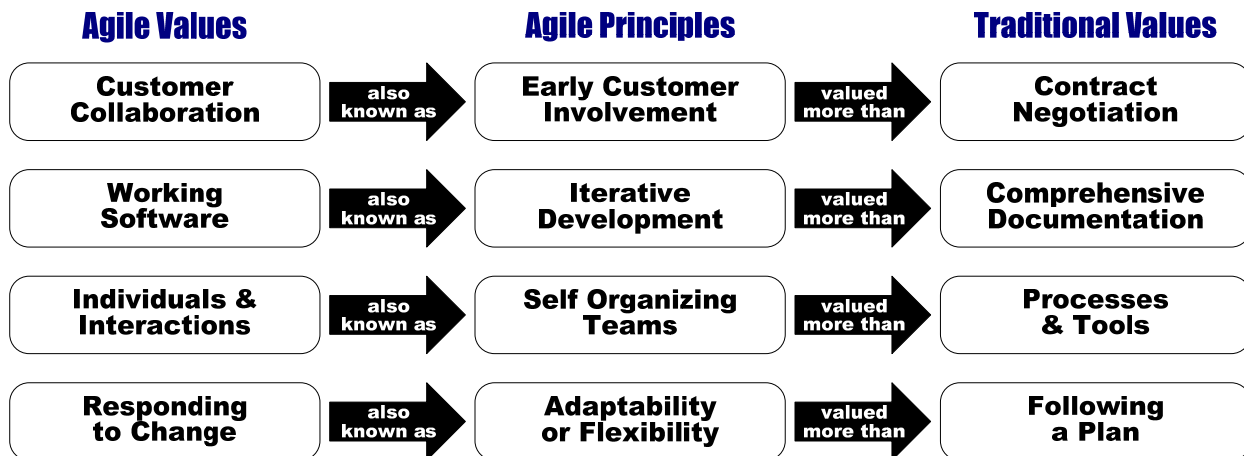


Figure 2. Four major values of agile methods (found in fields of innovation and new product development)

Agile methods are based on four broad values derived from the Agile Manifesto: (1) individuals and interactions, (2) customer collaboration, (3) working software, and (4) respond to change.² These four values are common to all major forms of agile methods such as Scrum, Extreme Programming, Feature Driven Development, Dynamic Systems Development, and Crystal Clear. They were developed and agreed to by the creators of all of these agile methods, who formed a council of 17 experts and met in 2001 in order to find a good alternative to traditional methods. Although they seem a little high-level and pie-in-the-sky, they are the defining characteristics of agile methods, what differentiates agile from traditional methods, and should not be overlooked. Some software engineers feel one isn't using an agile method unless one is using Scrum or Extreme Programming, or a specific practice like daily standup meetings or pair programming. There is some merit to these positions, so we do not want to discourage software engineers from following agile methods such as Scrum and Extreme Programming to the literal letter-of-the-law. Software engineers should not forget to draw upon the values of agile methods: communicate often, collaborate with customers, build working software, and be adaptable to changing needs. Agile methods are new product development processes and the four values of agile methods are not exclusive to agile methods, but common within the product innovation and marketing fields. Scholars are starting to realize that these values are more important than the individual practices.

Literature Review

Although software engineering is the third fastest growing career in the U.S., the number of new students pursuing computer science and software engineering degrees is in alarming decline.^{41,42} There was only one masters degree program in software engineering in the U.S. in the 1970s, a few dozen arose between 1995 and 2005, and only U.S. two bachelors programs exist in 2008.²³ By 2002, Agile Methods had swept the globe, as many as 67% of small to medium-sized firms were using them, and traditional methods were in sharp decline in the commercial marketplace. The earliest known academic research on agile methods was started in Harvard in the mid 1990s, but soon appeared at universities throughout Europe, Canada, and the Middle East around 2002. Much of this research was inspired by the University of Maryland's computer science program, which promoted validation of new software engineering theories by experimentation for decades.

Table 1. Studies of Agile Methods in Bachelors and Masters-level Degree Programs (from 2003 to 2008)

No.	Author(s)	Year	Tech.	Level	Course	Virtual	Issues	Method	N
1.	Dubinsky & Hazzan	2003	XP	B.S.	Capstone	No	Difficulty, Collaboration	Case	12
2.	Hedin, Bendix, & Magnusson	2003	XP	B.S.	2 nd year	No	Teamwork, Training	Case	107
3.	Keefe & Dick	2004	XP	M.S.	Capstone	No	Training, Time, Tools	Case	4
4.	Alfonso & Botia	2005	Agile	B.S.	Mixed	No	Teamwork, Difficulty	Case	153
5.	Hedin, Bendix, & Magnusson	2005	XP	M.S.	Mixed	No	Teamwork, Skill Level	Case	125
6.	Loftus & Ratcliffe	2005	XP	M.S.	Capstone	No	Teamwork, Skill Level	Case	18
7.	Marrington, Hogan, & Thomas	2005	Agile	B.S.	Mixed	No	Teamwork, Collaboration	Survey	35
8.	Melnik & Maurer	2005	XP	Mixed	Mixed	No	Adoption, Transfer	Survey	693
9.	Mendes, Al-Fakhri, & Luxton-Reilly	2005	PP	B.S.	2 nd year	No	Teamwork, Culture	Exp	300
10.	Janzen & Saiedian	2006	TDD	Mixed	1 st year	No	Adoption, Training	Mixed	64
11.	Laplante	2006	XP	M.S.	Capstone	Yes	Adoption, Teamwork	Case	72
12.	LeJeune	2006	XP	B.S.	Capstone	No	Design, Refactoring	Survey	18
13.	Smith, Stoecklin, & Serino	2006	Refactor	B.S.	1 st year	No	Documentation, Testing	Case	???
14.	Spacco & Pugh	2006	TDD	B.S.	1 st year	No	Adoption, Motivation	Case	34
15.	Briggs & Girard	2007	TDD	B.S.	1 st year	No	Difficulty, Volume	Case	???
16.	Kessler & Dykman	2007	Crystal	B.S.	2 nd year	No	Difficulty, Volume	Case	105
17.	Miller & Smith	2007	TDD	B.S.	3 rd year	No	Difficulty, Consistency	Case	135
18.	Stoecklin, Smith, & Serino	2007	Refactor	B.S.	1 st year	No	Criteria, Evidence	Case	???
19.	Jacobson & Schaefer	2008	PP	B.S.	1 st year	No	Teamwork, Reliability	Case	175
20.	Janzen & Saiedian	2008	TDD	B.S.	1 st year	No	Student Reluctance	Exp	140
21.	Kollanus & Isomottonen	2008	TDD	M.S.	4 th year	No	Training, Difficulty	Survey	52
22.	Murphy, Phung, & Kaiser	2008	XP	M.S.	2 nd year	Yes	Communication, Tools	Case	90
23.	Simon & Hanks	2008	PP	B.S.	1 st year	No	Scheduling	Survey	11
24.	Stapel, Lubke, & Knauss	2008	XP	M.S.	2 nd year	No	Time Availability	Case	48
25.	Tan, Tan, & Teo	2008	Agile	B.S.	3 rd year	No	Time Availability	Case	600

Common patterns associated with introducing agile methods into bachelors and masters-level degree coursework seemed to emerge based on an analysis of 25 studies (as shown in Table 1). Most of these studies introduced agile methods into bachelors degrees and over half introduced an entire agile method versus practices such as pair programming or test driven development. Nearly half of these studies used Extreme Programming for teaching agile methods, the majority were not virtual classrooms, and only 20% used agile methods as part of a final capstone course. These are only a fraction of the universities in the world, because most universities do not offer software engineering courses and existing programs may be entrenched in traditional methods. Often cited issues associated with incorporating agile methods into a single semester worth of coursework included teamwork, difficulty, adoption, training, time availability, and workload. By and large, no training was provided, resistance to change was common, working in teams was problematic, customer interaction was difficult, and a lot of administrative effort was required.

Conceptual Model

Although most if not all major universities around the world offer bachelors, masters, and even doctoral degrees in computer science, far fewer universities offer software engineering degrees. Typical computer science courses may include several computer programming languages, data structures, algorithms, digital architecture, operating systems, databases, statistics, and calculus. Typical software engineering courses may include project management, analysis, architecture, construction, verification, validation, maintenance, configuration control, and quality assurance. However, there is no typical curriculum for a course on agile methods outside of the practices associated with the various kinds of agile methods such as Scrum,³³ Extreme Programming,⁵ etc. Therefore, a conceptual model was developed for the capstone course of a masters degree in software engineering based on the release planning methodology from Extreme Programming.⁶ In actuality, we just completed a four-year study of agile methods, published our results in multiple magazines and textbooks, and analyzed the design of a multitude of agile methods.²⁸⁻³² We developed introductory training, devised release planning templates, compiled bibliographic references, obtained formal training, and solicited feedback from both academia and industry.

Table 2. Conceptual Model for a Capstone Course on Software Engineering (based on Agile Methods)

Week	Dates	Phase	Deliverables	Software Engineering Discipline
1	09/10 - 09/14	Initiation	<ul style="list-style-type: none"> • Resumes • Project team 	<ul style="list-style-type: none"> • Project management • Project management
2	09/15 - 09/21		<ul style="list-style-type: none"> • Agile introduction • Project charter • Project scope 	<ul style="list-style-type: none"> • Project management • Project management
3	09/22 - 09/28		<ul style="list-style-type: none"> • User stories • System metaphor • Release plan 	<ul style="list-style-type: none"> • Requirements engineering • System architecture/design • Project management
4	09/29 - 10/05	Iteration 1	<ul style="list-style-type: none"> • Development tasks • Iteration plan 	<ul style="list-style-type: none"> • Project management • Project management
5	10/06 - 10/12		<ul style="list-style-type: none"> • Unit tests • Acceptance tests 	<ul style="list-style-type: none"> • Verification and validation • Verification and validation
6	10/13 - 10/19		<ul style="list-style-type: none"> • Peer evaluation • Customer satisfaction • Lessons learned 	<ul style="list-style-type: none"> • Project management • Project management
7	10/20 - 10/26	Iteration 2	<ul style="list-style-type: none"> • Development tasks • Iteration plan 	<ul style="list-style-type: none"> • Project management • Project management
8	10/27 - 11/02		<ul style="list-style-type: none"> • Unit tests • Acceptance tests 	<ul style="list-style-type: none"> • Verification and validation • Verification and validation
9	11/03 - 11/09		<ul style="list-style-type: none"> • Peer evaluation • Customer satisfaction • Lessons learned 	<ul style="list-style-type: none"> • Project management • Project management
10	11/10 - 11/16	Iteration 3	<ul style="list-style-type: none"> • Development tasks • Iteration plan 	<ul style="list-style-type: none"> • Project management • Project management
11	11/17 - 11/23		<ul style="list-style-type: none"> • Unit tests • Acceptance tests 	<ul style="list-style-type: none"> • Verification and validation • Verification and validation
12	11/24 - 11/30		<ul style="list-style-type: none"> • Peer evaluation • Customer satisfaction • Lessons learned 	<ul style="list-style-type: none"> • Project management • Project management
13	12/01 - 12/07	Closeout	<ul style="list-style-type: none"> • Final presentation • Paper 	<ul style="list-style-type: none"> • Project management • Project management

Research Method

Agile methods were introduced as part of the capstone course towards a masters degree in software engineering, which was the 16th capstone course offered over a period of eight years. The capstone course was the final course as part of a measured program in software engineering in which the students were given a healthy dose of traditional methods over a two year period. The program included courses on project management, systems engineering, analysis, design, verification and validation, and maintenance, among other technical and managerial electives. The program was centered around the Software Engineering Body of Knowledge, which is based on 12 seven-page knowledge areas ranging from topics in requirements analysis to maintenance.¹ The Software Engineering Body of Knowledge represents the sum of knowledge and experience gained about software engineering over the 65 years since the advent of the electronic computer.

Project Description
<i>Each team will develop a general-purpose business-to-consumer (B2C) class electronic commerce website for buying and selling digital media products. The end-product should be an electronic retailing website or storefront similar to Amazon.com where consumers can search for products and identify and purchase the products they desire. One constraint is that the website will be for digital media products, such as e-books, market studies, white papers, company briefings, and other forms of digital media often in PDF format. The website should have features and functions for both the buyer and the seller. For instance, the buyer should have access to a catalog of products, be able to browse through or search for products, get detailed information on products, add items of interest to a shopping cart, create user accounts with personal information, order products using credit cards, and obtain customer services such as rating transaction quality, reporting problems, or resolving account disputes. The seller should also have features and functions, such as an attractive website that presents a pleasant shopping experience that encourages customers to shop for products using this website. Furthermore, the seller should be able load products into a catalog, including photos, product descriptions, prices, sale items, special deals, and other promotional aids in order to obtain business.</i>

Figure 3. Project description for a capstone course (towards a masters degree software engineering)

Agile methods were selected as the main software engineering method for this capstone course, which are considered antithetical to principles in the Software Engineering Body of Knowledge. One goal was to introduce contemporary software engineering approaches such as agile methods and to challenge the students to use the Software Engineering Body of Knowledge as necessary. The purpose of the capstone course was for the students to use software engineering principles to complete a major team project using agile methods as the main software development life cycle. Fifteen people enrolled in the capstone course and self-organized into three teams of five people, most of which were in their final course towards their degree, with the exception of one student. The students were also provided with a detailed syllabus concisely outlining the specific course requirements along with an unambiguous list of deliverables, delivery dates, and expectations. The students were provided with Microsoft Visual Studio .NET, Visio, and Project, and were instructed to “build-versus-buy,” consistent with the Software Engineering Body of Knowledge. The students were not provided with formal training, tools, or technologies in agile methods as part of their challenge to draw upon their education and experience to deal with a sudden change. The students were given a high-level project description, assigned a customer who had training, knowledge, and experience with agile methods, and provided with an initial list of user stories.⁸ The program’s director, visiting faculty member, and teaching assistant administered the course.

Demographics

The 15 students averaged about 11 years of software development experience and completed educational degrees in areas such as computer science, electrical engineering, and architecture. Most of the students were advancing in their careers to such roles as project managers, technical leads, systems analysts, and other software development roles other than full-time programming. However, their technical skills included ASP, .NET, Basic, C, C#, C++, J2EE, Java, Javascript, Lotus Notes and Script, Pascal, Perl, PHP, Python, Ruby, SQL, Visual Basic, Unix, and Linux. About half of the students were working on U.S. DoD and other government software projects and the other half were working on small to medium-sized projects in the commercial industry. The large majority of the students had a steady exposure to the complete software lifecycle and was comfortable with traditional methods and the Software Engineering Body of Knowledge. Two or three of the 15 students had either heard of or used agile methods such as Scrum and Extreme Programming to some degree, but agile methods were new to the majority of students. The first team to form was Awesomesauce, they were the most cohesive group, they had the most programming experience, and up to three of them were involved in coding for this project. The second team to form was Kestrel, they were moderately cohesive, they had the second most programming experience, and up to two of them were regularly involved in coding this project. The third team to form was Yellowstone, they had the most experience of the three teams, they had the least current programming experience, and one member did the majority of the coding. Traditional methods and the Software Engineering Body of Knowledge seemed to be extremely well-suited for the transition of the students into the advanced stages of most of their careers.

Course Administration

There were three people involved in the administration of UMUC's capstone course: the director of the software engineering program, a visiting faculty member, and a regular teaching assistant. The director served as the visionary for the course and wanted to both introduce agile methods to the students as well as challenge them to apply the Software Engineering Body of Knowledge. The visiting faculty member was responsible for designing the course, serving as subject matter expert on agile methods, serving as customer, and analyzing each team's technical performance. The teaching assistant was responsible for organizing the administrative details and syllabus, and managing day-to-day course operations and communications between the faculty and students. The basic course was designed in about 40 hours using the release planning methodology from Extreme Programming as a basis (e.g., user stories, metaphors, release plans, iteration plans, etc.) About a month was needed to evaluate training materials on agile methods available on the web, design introductory training, collect reference material, and support other administrative tasks. The visiting faculty member spent another three weeks coaching the teams during the initiation phase in order to adapt their understanding of traditional methods to concepts in agile methods. Once the iterations got underway, the visiting faculty member served as the customer to supply user stories, acceptance criteria, routine coaching in agile methods, and administrative support. The visiting faculty member spent about six hours providing support during the first iteration, about 12 hours during the second iteration, and approximately 20 hours during the third iteration. This was in addition to fielding numerous email messages between the faculty, staff, teams, and individual students, which were mostly to establish telephone conversations and teleconferences.

Data Collection

The initial capstone project phase started out a little chaotic as the students began exploring agile methods after being steeped in the Software Engineering Body of Knowledge for two long years. The students asked the typical questions about the nature of the product they were to build, the technologies they were to use, who the customer was, and the deliverables they had to produce. Weekly questions were posted to help gauge student feelings about agile methods, which yielded big concerns such as, “I hate it,” “It’s scary,” and “It’s what’s wrong with Western Civilization.” We responded by posting responses addressing such issues as agile versus traditional methods, user stories, metaphors, hybrids, customer roles, virtual teams, documentation, and maintenance. We had to jumpstart the teams by scheduling customer meetings, explaining the project scope, providing initial user stories, and answering general questions, which seemed to calm the storm. (See Table 3 for a detailed analysis of team performance on an iteration-by-iteration basis.)

Team Awesomesauce. The first team to form was a somewhat of an enigma as they initially exhibited a high degree of initiative, and then had to be prodded into contacting the customer. Awesomesauce came out of the gate stumbling as they struggled to combine Scrum and release planning, missing some key user stories, and filling their product backlog with technical tasks. Awesomesauce picked up the pace using VersionOne to automate Scrum, fixing their backlog, and implementing a complex user story consisting of an inventory database all in one fell swoop. By Iteration 2, Awesomesauce was aggressively implementing user stories and demonstrated the ability to perform an end-to-end e-commerce transaction using PayPal for credit card processing. By the end of Iteration 3, Awesomesauce was rounding out its e-commerce website with lower priority user stories such as best-seller lists, newsletters, blogs, and a variety of embellishments.

Team Kestrel. The second team to form was much like the first team, initially exhibiting a high degree of initiative, but like the first, had to be prodded into contacting the customer to get going. Kestrel also came out of the gate stumbling, constantly referring to hybrid methods as a “salad bar” approach to agile methods, being unprepared at the initial meeting, and starting off slowly. Kestrel picked up the pace by using a Wiki to prioritize their user stories, and scheduling just-in-time meetings to review wireframes and prototypes with the customer before each final iteration. By Iteration 2, Kestrel was implementing user stories at a faster pace, and more complex ones at that, completing their inventory database and enabling customers to easily customize the layout. By the end of Iteration 3, Kestrel was rounding out its e-commerce website with lower priority user stories such as a database-driven best-sellers list and PayPal for credit card processing.

Team Yellowstone. The third team refused to use the university’s institutional collaboration tool and appeared to struggle a great deal, although they were the first to contact the customer. Yellowstone came out of the gate quickly, setting up a web server before the first iteration, and designing a rapid prototype before the first customer meeting and initial user stories were given. Yellowstone quickly set up a Wiki for tracking user stories, implemented an inventory database, which was a complex user story, and engaged in a series of rich interactions with the customer. At that point Yellowstone hit a brick wall as the lead developer felt he was doing all of the work, necessitating the use of virtual pair programming sessions to share the developmental workload. By the end of Iteration 2, Yellowstone was functioning as a team, completed its features to allow website customization, and implemented shopping carts and other core features by Iteration 3.

Table 3. Analysis of Detailed Team Performance Data for a Capstone Course (based on Agile Methods)

Phase	Team	Strength	Weakness
Initiation	Awesomesauce	<ul style="list-style-type: none"> • Got off to a fast start. • Formed their team first. • Elected to use Scrum. 	<ul style="list-style-type: none"> • Standoffish towards customer. • Failed to implement release planning. • Slow to begin development activities.
	Kestrel	<ul style="list-style-type: none"> • Captured all initial user stories. • Suggested new user stories. • Elected to use Scrum. 	<ul style="list-style-type: none"> • Hesitant to engage customer. • Unprepared during initial meetings. • Slow to begin development activities.
	Yellowstone	<ul style="list-style-type: none"> • First to engage customer. • Augmented user stories with wireframes. • First to develop operational website. 	<ul style="list-style-type: none"> • Too much early focus on coding. • Not enough focus on user stories. • Not listening to the customer enough.
Iteration 1	Awesomesauce	<ul style="list-style-type: none"> • Used agile methods workflow tool. • Implemented complex user story first. • Internal teamwork and collaboration. 	<ul style="list-style-type: none"> • Hesitant to engage customer. • Rough transition to release planning. • Rough usability and aesthetic design.
	Kestrel	<ul style="list-style-type: none"> • Good adaptation of release planning. • Used Wiki for customer collaboration. • Used just-in-time customer interaction. 	<ul style="list-style-type: none"> • Slow start with simple user stories. • Rough usability and aesthetic design. • Not enough people developing code.
	Yellowstone	<ul style="list-style-type: none"> • Established good customer relations. • Used many collaborative technologies. • Aggressively implemented user stories. 	<ul style="list-style-type: none"> • Implemented unwanted user stories. • Rough usability and aesthetic design. • Not enough people developing code.
Iteration 2	Awesomesauce	<ul style="list-style-type: none"> • More frequent customer interaction. • Aggressively implemented user stories. • First to begin completing website. 	<ul style="list-style-type: none"> • Deferred design decisions to customer. • Rough usability and aesthetic design. • Not enough people developing code.
	Kestrel	<ul style="list-style-type: none"> • Proactive attention to user story detail. • Used Wiki for customer collaboration. • Used just-in-time customer interaction. 	<ul style="list-style-type: none"> • Use of wireframes instead of coding. • Rough usability and aesthetic design. • Not enough people developing code.
	Yellowstone	<ul style="list-style-type: none"> • Used virtual pair programming. • Used more collaborative technologies. • Good internal teamwork/collaboration. 	<ul style="list-style-type: none"> • Slower programming velocity. • Rough usability and aesthetic design. • Not enough people developing code.
Iteration 3	Awesomesauce	<ul style="list-style-type: none"> • Frequent customer interaction. • Aggressively implemented user stories. • Most complete end-to-end website. 	<ul style="list-style-type: none"> • Little interpersonal customer relations. • Rough usability and aesthetic design. • Not enough people developing code.
	Kestrel	<ul style="list-style-type: none"> • Very good customer interaction. • Used better collaborative technologies. • Almost complete end-to-end website. 	<ul style="list-style-type: none"> • Drilled too deep rather than broadly. • Rough usability and aesthetic design. • Not enough people developing code.
	Yellowstone	<ul style="list-style-type: none"> • Strong interpersonal customer trust. • Good use of collaborative technology. • Good balance of process vs. product. 	<ul style="list-style-type: none"> • Slowest programming velocity. • Rough usability and aesthetic design. • Not enough people developing code.
Closeout	Awesomesauce	<ul style="list-style-type: none"> • Demonstrated grasp of Scrum. • Held high regard for agile methods. • Good team turnout for final briefing. 	<ul style="list-style-type: none"> • Final briefing was narrowly focused • Didn't demonstrate final website. • Mildly adversarial towards customer.
	Kestrel	<ul style="list-style-type: none"> • Comprehensive final briefing. • Exhibited good internal teamwork. • Identified software engineering issues. 	<ul style="list-style-type: none"> • Didn't speak to agile methods values. • Didn't demonstrate final website. • Didn't implement virtual team very well.
	Yellowstone	<ul style="list-style-type: none"> • Good discussion of pair programming. • Discussed collaborative technologies. • Discussed development technologies. 	<ul style="list-style-type: none"> • Exhibited underlying individualism. • Didn't demonstrate final website. • Final briefing was not comprehensive.
Overall	Awesomesauce	<ul style="list-style-type: none"> • Grasped technical aspects of Scrum. • Reached optimal coding velocity. • Used agile methods workflow tool. 	<ul style="list-style-type: none"> • Didn't grasp key agile methods values. • Rough usability and aesthetic design. • Didn't use collaborative technologies.
	Kestrel	<ul style="list-style-type: none"> • Demonstrated grasp of agile methods. • Reached a good coding velocity. • Used some collaborative technologies. 	<ul style="list-style-type: none"> • Didn't grasp hybrid methods concept. • Rough usability and aesthetic design. • Used few collaborative technologies.
	Yellowstone	<ul style="list-style-type: none"> • Good balance of process vs. product. • Great use of collaborative technology. • Good grasp of agile methods values. 	<ul style="list-style-type: none"> • Stumbled implementing teamwork. • Rough usability and aesthetic design. • Sort of ran out of steam in the end.

Data Analysis

This was more of a case study, rather than an experiment with a control group consisting of measurements and statistics, so our outcomes were mostly qualitative rather than quantitative. Only the principal faculty member knows whether this capstone course was successful or not, because he administered the previous 15 capstone courses and has eight years of empirical data. If the measure of success is an operational software product that satisfies customer requirements, then one might say that this capstone course was one of the most successful of the prior courses. Prior students were rewarded for their ability to produce software engineering documentation, while the students in this course were rewarded for a balance of just-enough process and product. If quality is defined as conformance to customer requirements, then all of the teams satisfied the requirements and defect density was zero defects per thousand lines of code (or function point). We could have collected traditional measures such as effort, cost, size, defects, reliability, etc., but metrics and models better suited to the values of agile methods would have been much better. For example, customer satisfaction, teamwork, interpersonal trust, team cohesion, adaptability to change, communication quality, number of working software iterations, project success, etc.

The teams used agile methods to exercise disciplined software engineering principles such as project management, requirements engineering, design, verification, validation, and many others. The release planning methodology from Extreme Programming consists of a rather thorough and complete project management approach with the right balance flexibility and process discipline. User stories also serve as a robust mechanism to capture customer requirements and form the basis for a disciplined requirements engineering process ideally suited for customer interaction. Agile methods serve as a thorough, end-to-end verification and validation process, and quality control is not just limited to test first development (unit testing) as traditionalists errantly claim. Customers are involved throughout the project, and provide requirements that are verified and validated at frequent intervals rather than at the end when it is too late and expensive to change. If customers are involved in a project to just the right degree, then requirements are expressed, captured, implemented, verified, and validated on a daily, weekly, and even bi-weekly basis. Executing agile methods well may be one of the best forms of verification and validation, or even quality control and discipline, which few traditional methods have been able to achieve.

Agile methods form the basis of the right balance of just-enough software engineering discipline, so that development teams can focus on technology, product development, and software design. Agile methods come with project management, requirements engineering, software architecture, documentation, and even software maintenance without a separate process for product evolution. Agile methods also need organizational commitment, resources, and training to succeed, as well as tools for project management, process management, testing, and configuration management. However, agile methods are more about its core values of customer collaboration, teamwork, working software, and flexibility, rather than manufacturing measures such as defect density. Teams that had better programming skills, more programmers, better teamwork, and knowledge of agile methods such as Scrum had better productivity and quality than other less capable teams. At first glance, it seems as though teams with better technical skills and just enough customer interaction were more successful than teams that only had extremely good customer interaction. Teams with a good balance of just-enough process and product discipline also performed better than teams who used numerous virtual collaboration tools, technologies, and techniques.

Conclusion

The Software Engineering Body of Knowledge may be regarded as the evolution of scientific management principles over the last century, which in part seeks to objectify human activity. However, contemporary software engineering principles, especially those within agile methods, are all about individuals and interactions between teams as well as collaboration with customers. People within software development and customer teams are not automatons or machines, and cannot be subjected to Tayloristic reductionism and objectified by processes and documentation. Teamwork and customer collaboration are the basic values of agile methods and may be missing within traditional methods. With this in mind, some of the lessons we learned, included:

- Training. Agile methods training should be mandatory before introducing them in software engineering capstone courses. Students should be provided with basic training through earlier coursework or commercial training providers. Faculty and staff should be trained as well.
- Teamwork. Teamwork training should be mandatory before introducing agile methods. Students are penalized for collaborating in computer science, but software engineering and agile methods courses require teamwork. Faculty and staff need teamwork training too.
- People skills. Training in people skills should be mandatory for faculty, staff, and students. Traditional methods are based on the principles of scientific management, in which software development activities are objectified. However, people can't and shouldn't be objectified.
- Virtual teams. Training in virtual teams should also be mandatory. Software engineering education often takes place in a virtual environment. Use periodic face-to-face interactions to augment virtual projects in order to enhance communication, collaboration, and trust.
- Collaborative tools. Provide a rich variety of collaborative tools for virtual teams. A Wiki is simply not enough. Virtual software engineering teams using agile methods need tools such as WebEx and Skype. Traditional media can also be used enhance virtual teams.
- Workflow tools. Provide a rich variety of workflow tools specially designed for agile methods. Do not leave these to chance. The best agile methods are hybrids and choosing the wrong tool can thwart optimal performance. Use tools that support hybrid agile methods.
- Developmental tools. Not only provide teams using agile methods with contemporary software development tools, but provide them with training as well. Don't just provide tools with a free academic license (if the students haven't been trained to use them for some time).
- Aesthetic design. Place a greater emphasis on aesthetic design. Poor aesthetic design costs trillions of dollars in lost revenues each year. Software engineering curriculums need to place a greater emphasis on aesthetic design, not just software engineering documentation.
- Buy vs. build. Place a greater emphasis on buy-versus-build. This is a knowledge or process area that has been ignored by software engineering and process improvement standards. Software engineering students should be taught to buy before building as a rule-of-thumb.

One assumption made when designing this capstone course for a masters degree in software engineering was that agile methods should be objectified using the lens of traditional methods. After more than four years of intensive research resulting in numerous publications, we are now coming to the gradual realization that agile methods are more about people versus process skills. We could argue that our capstone course was successful, because three teams learned a lot about agile methods, successfully applied them, and built three fully functional e-commerce websites. On the other hand, we could argue that we are discovering agile methods for the very first time.

References

1. Abran, A., Moore, J. W., Bourque, P., & Dupuis, R. (2004). *Guide to the software engineering body of knowledge*. Los Alamitos, CA: IEEE Computer Society.
2. Agile Manifesto. (2001). *Manifesto for agile software development*. Retrieved January 1, 2009, from <http://www.agilemanifesto.org>
3. Alfonso, M. I., & Botía, A. (2005). An iterative and agile process model for teaching software engineering. *Proceedings of the 18th Conference on Software Engineering Education and Training (CSEET 2005), Ottawa, Canada*, 9-16.
4. Anonymous. (2007). The 2007 software 500 ranking: Listing of revenue, growth, and business sector. *Software Magazine*, 26(5), 27-57.
5. Beck, K. (2001). *Extreme programming: Embrace change*. Upper Saddle River, NJ: Addison-Wesley.
6. Beck, K., & Fowler, M. (2001). *Planning extreme programming*. Upper Saddle River, NJ: Addison-Wesley.
7. Briggs, T., & Girard, C. D. (2007). Tools and techniques for test driven learning in CS1. *Journal of Computing Sciences in Colleges*, 22(3), 37-43.
8. Cohn, M. (2004). *User stories applied: For agile software development*. Boston, MA: Addison-Wesley.
9. Dubinsky, Y., & Hazzan, O. (2003). Extreme programming as a framework for student project coaching in computer science capstone courses. *Proceedings of the First IEEE International Conference on Software, Science, Technology, and Engineering (SWSTE 2003), Herzelia, Israel*, 53-59.
10. Hedin, G., Bendix, L., & Magnusson, B. (2003). Introducing software engineering by means of extreme programming. *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003), Portland, Oregon*, 586-593.
11. Hedin, G., Bendix, L., & Magnusson, B. (2005). Teaching extreme programming to large groups of students. *Journal of Systems and Software*, 74(2), 133-146.
12. Highsmith, J. A. (2002). *Agile software development ecosystems*. Boston, MA: Addison Wesley.
13. Jacobson, N., & Schaefer, S. K. (2008). Pair programming in CS1: Overcoming objections to its adoption. *SIGCSE Bulletin*, 40(2), 93-96.
14. Janzen, D. S., & Saiedian, H. (2006). Test driven learning: Intrinsic integration of testing into the CS/SE curriculum. *Proceedings of the 37th ACM Technical Symposium on Computer Science Education (SIGCSE 2006), Houston, Texas, USA*, 254-258.
15. Janzen, D. S., & Saiedian, H. (2008). Test driven learning in early programming courses. *Proceedings of the 39th ACM Technical Symposium on Computer Science Education (SIGCSE 2008), Portland, Oregon, USA*, 532-536.
16. Keefe, K., & Dick, M. (2004). Using extreme programming in a capstone project. *Proceedings of the Sixth Australasian Computing Education Conference (ACE 2004), Dunedin, New Zealand*, 151-160.
17. Kessler, R., & Dykman, N. (2007). Integrating traditional and agile processes in the classroom. *Proceedings of the 38th ACM Technical Symposium on Computer Science Education (SIGCSE 2007), Covington, Kentucky, USA*, 312-316.
18. Kollanus, S., & Isomottonen, V. (2008). Test driven development in education: Experiences with critical viewpoints. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2008), Madrid, Spain*, 124-127.
19. Laplante, P. (2006). An agile graduate software studio course. *IEEE Transactions on Education*, 49(4), 417-419.
20. LeJeune, N. F. (2006). Teaching software engineering practices with extreme programming. *Journal of Computing Sciences in Colleges*, 21(3), 107-117.
21. Loftus, C., & Ratcliffe, M. (2005). Extreme programming promotes extreme learning? *Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2005), Monte de Caparica, Portugal*, 311-315.

22. Marrington, A., Hogan, J. M., & Thomas, R. (2005). Quality assurance in a student based agile software engineering process. *Proceedings of the 16th Australian Software Engineering Conference (ASWEC 2005), Brisbane, Australia*, 324-331.
23. Mead, N., Carter, D., & Lutz, M. (1997). *The state of software engineering education and training*. IEEE Software, 14(6), 22-25.
24. 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 (ICSE 2005), St. Louis, Missouri, USA*, 481-488.
25. Mendes, E., Al-Fakhri, L. B., & Luxton-Reilly, A. (2005). Investigating pair programming in a 2nd year software development and design computer science course. *Proceedings of the 10th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2005), Monte de Caparica, Portugal*, 296-300.
26. Miller, J., & Smith, M. (2007). A TDD approach to introducing students to embedded programming. *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2007), Dundee, Scotland, United Kingdom*, 33-37.
27. Murphy, C., Phung, D., & Kaiser, G. (2008). A distance learning approach to teaching extreme programming. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2008), Madrid, Spain*, 199-203.
28. Rico, D. F. (2007). Effects of agile methods on electronic commerce: Do they improve website quality? *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), Waikaloa, Big Island, Hawaii*.
29. Rico, D. F. (2008). Effects of agile methods on website quality for electronic commerce. *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008), Waikaloa, Big Island, Hawaii*.
30. Rico, D. F. (2008). What is the ROI of agile vs. traditional methods? An analysis of extreme programming, test-driven development, pair programming, and scrum (using real options). *TickIT International*, 10(4), 9-18.
31. Rico, D. F., Sayani, H. H., & Field, R. F. (2008). History of computers, electronic commerce, and agile methods. In M. V. Zelkowitz (Ed.), *Advances in computers: Emerging technologies, Vol. 73*. San Diego, CA: Elsevier.
32. Rico, D. F., Sayani, H. H., Stewart, J. J., & Field, R. F. (2007). A model for measuring agile methods and website quality. *TickIT International*, 9(3), 3-15.
33. Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*. Upper Saddle River, NJ: Prentice-Hall.
34. Simon, B., & Hanks, B. (2008). First year student's impressions of pair programming in CS1. *ACM Journal on Educational Resources in Computing*, 7(4), 5:1-5:28.
35. Smith, S., Stoecklin, S., & Serino, C. (2006). An innovative approach to teaching refactoring. *Proceedings of the 37th ACM Technical Symposium on Computer Science Education (SIGCSE 2006), Houston, Texas, USA*, 349-353.
36. Spacco, J., & Pugh, W. (2006). Helping students appreciate test driven development (TDD). *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2006), Portland, Oregon, USA*, 907-913.
37. Stapel, K., Lubke, D., & Knauss, E. (2008). Best practices in extreme programming course design. *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany*, 769-775.
38. Stoecklin, S., Smith, S., & Serino, C. (2007). Teaching students to build well formed object oriented methods through refactoring. *Proceedings of the 38th ACM Technical Symposium on Computer Science Education (SIGCSE 2007), Covington, Kentucky, USA*, 145-149.
39. Tan, C. H., Tan, W. K., & Teo, H. H. (2008). Training students to be agile information systems developers: A pedagogical approach. *Proceedings of the Seventh Joint SIGMIS/SIGCPR Conference and Doctoral Student Consortium (SIGMIS-CPR 2008), Charlottesville, Virginia, USA*, 88-96.
40. U.S. Census Bureau. (2008). *E-stats: E-commerce 2006*. Washington, DC: Author.
41. United States Department of Labor. (2007). *Employment projections: 2006 to 2016*. Washington, D.C.: Author.
42. Zweben, S. (2008). 2006-2007 taulbee survey: Ph.D. production exceeds 1,700, undergraduate enrollment trends still unclear. *Computing Research News*, 20(4), 6-17.