Use of Agile Methods in Software Engineering Education

Dr. David F. Rico
Boeing,
dave1@davidfrico.com

Dr. Hasan H. Sayani
University of Maryland University College
hsayani@umuc.edu

Abstract—The use of contemporary software development approaches such as agile methods is growing in widespread use throughout the world. Although some universities are starting to teach them, courses on agile methods at the undergraduate and graduate levels are still a new phenomenon. The University of Maryland University College (UMUC) adapted agile methods for its capstone course towards a master’s degree in software engineering in the Fall of 2008. Three distributed teams of five students were asked to use agile methods to build competing electronic commerce websites. With little training in agile methods, virtual teams, collaboration tools, or web design, each of the three teams completed fully functional e-commerce websites using agile methods in little more than 13 weeks. Teams who struck an optimum balance of customer collaboration, use of agile methods, and technical programming ability had better productivity and website quality.

I. SOFTWARE ENGINEERING EDUCATION

The term software engineering was coined in 1968 as a result of the software crisis or demand for software by the U.S. military along with difficulty of fulfilling that demand. Computer programmers began to be referred to as software engineers from 1970 to 1980, especially among U.S. DoD firms, and a handful of leading-edge colleges and universities began offering master’s degrees in software engineering. Software engineering became the third fastest growing career field in the U.S. from 1995 to 2005. Much of this growth coincided with the rise of the commercial shrink-wrapped software industry, the spread of powerful, but low-cost personal computers, and the rise of the Internet. Non-traditional colleges and universities seemed to spring up overnight and master’s degrees in software engineering became available to the masses by the late 1990s. Traditional universities continued offering bachelor’s and master’s degrees in computer science, while non-traditional, often online, universities gravitated towards master’s degrees in software engineering.

II. IMPETUS FOR CHANGE

UMUC, with 170,000 full and part-time students, undertook the challenge of teaching agile methods as part of its capstone course towards a master’s degree in software engineering. The stewards of UMUC’s master’s degree program in software engineering have always tried to stay on the leading edge of the computer programming industry in order to provide its students with marketable skills. It does so by requiring students to apply the latest computer programming technologies as they emerge, such as Java, Perl, C#, .NET, ASP, Ruby, Linux, etc. The students are often encouraged to use the latest open source software innovations, such as Iona Fuse and Apache Service Mix for building service oriented architecture (SOA) and web services-based applications. Along the way, the students are also introduced to the disciplines associated with the Software Engineering Body of Knowledge (SWEBoK). These include requirements analysis, software architecture and design, coding standards, verification, validation, and maintenance.

The students are often encouraged to document the various software life cycle artifacts using as many IEEE software engineering standards as possible. From this standpoint, one might say that UMUC’s graduate software engineering degree program is based on traditional methods. However, true to its form, UMUC recognizes that agile methods are entering into mainstream use more and more, so they decided to infuse agile methods into its program in the Fall of 2008. Agile methods were introduced as part of the final, capstone course towards a master’s degree in software engineering. The students were expected to be highly-skilled and highly-competent computer programmers, in line with UMUC’s vision of what software engineering is supposed to be, a professional computer programming discipline. This was the 18th capstone course offered by UMUC, where students were to draw upon IEEE and SWEBoK disciplines to develop a software system for real customers using traditional methods. Only this time, UMUC’s software engineering students were to apply agile methods for their final capstone project instead.
III. COURSE DESIGN

In the Summer of 2008, two months before the capstone course was to begin, UMUC adapted agile methods for its graduate software engineering degree program. Institutional researchers who had been experimenting with agile methods and publishing their results were asked to adapt agile methods and design the lesson plan for the capstone course. The students would be allowed to use the agile method of their choice, such as Scrum, Extreme Programming, Feature Driven Development, Crystal Methods, Dynamic Systems Development, etc. However, they would be required to use the Release Planning methodology from Extreme Programming as a project management framework. The students were required to produce user stories, metaphors, release plans, iteration plans, development tasks, unit tests, and acceptance tests. This is not at all unusual, since Release Planning is a common practice used by both Extreme Programming and Scrum teams. Release Planning templates were created for the students to use, introductory training and other background materials were made available to the students, and an expert in agile methods was assigned to the students for day-to-day coaching and mentoring, seven days a week.

IV. DEMOGRAPHICS

Fifteen students signed up for the final capstone course towards a master’s degree in software engineering. The students averaged 11 years of experience and were proficient in the use of ASP, .NET, C#, C++, J2EE, Java, Javascript, Perl, PHP, Python, Ruby, SQL, Visual Basic, and Linux. The students had undergraduate degrees in computer science, information systems, and other technical disciplines. Most of the students had full-time jobs as software engineers and were already progressing beyond the early stages of their computer programming careers to such roles as project managers and software analysts. Half of the students worked on large U.S. government projects and the other half worked on small to medium-sized projects for commercial clients. All of the students had completed the coursework towards a master’s degree in software engineering at UMUC, save the capstone. They were all proficient in the use of traditional methods, the SWEBOK, and application of IEEE standards for documenting requirements, architectures, designs, code, and various others software life cycle artifacts. One or two of the students were familiar with agile methods, although none were currently using them. The large majority of the students were hearing of agile methods for the first time.

V. COURSE KICKOFF

The students were asked to self-organize into three teams of five computer programmers. Each of the teams was asked to build a fully operational and functional e-commerce website for buying and selling digital media products very much like Amazon.com. They were also asked to use agile methods instead of traditional methods for doing so. The use of agile methods was a little antithetical to everything they’d been taught during the prior two years of coursework. Historically, they’d been taught to develop requirements specifications, software architecture and design plans, test plans, and a plethora of documents such as software user and maintenance manuals. Many of these students had been promoting the use of traditional methods on their own real-world jobs and enjoyed doing so immensely. The notion of using agile methods for managing the software process was shocking, surprising, and most of all, confusing. They were befuddled as to which agile method to choose, use, and apply. The three teams settled down quite a bit once routine interactions with the customers, coaches, and mentors began. Each of the teams had exactly 13 weeks in order to complete their task, which included startup, execution, and closeout of their project. They were given three weeks to teach themselves as much as possible about agile methods and were assigned a customer, coach, and mentor. Then, the teams were instructed to build their e-commerce websites in three iterations.

VI. PROJECT DESCRIPTION

Each of the student teams was asked to develop an e-commerce website. A single customer was assigned to each of the three teams. Historically, the students were asked to find their own customers. This led to several problems, such as finding a customer, getting adequate requirements, and maintaining a consistent level of customer involvement throughout the capstone course. Since our primary objective was to have the teams focus on the use of agile methods, and customer experience with agile methods is imperative to project success, we simply assigned teams a single customer who was already familiar with agile methods. Furthermore, we had the customer give a single set of user stories to each of the three teams to minimize the effort for the customer and use this as a basis for having the teams build competing e-commerce websites. There was some concern that the students would build identical websites, but we reasoned that the students would build highly unique products, which they did. The following project and product description was given to each of the three teams:
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 purchase 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.

VII. COURSE EXECUTION

The director of UMUC’s software engineering program, a visiting faculty member, and a teaching assistant developed the course and provided oversight. It was the director’s idea to modernize UMUC’s software engineering program by infusing agile methods into the capstone course. The visiting faculty member designed the course, developed all of the preparatory materials, and served as coach, mentor, and customer. The teaching assistant provided administrative oversight. The visiting faculty member designed the basic course in about a week, gathered and organized background materials in about a month, and provided training for three weeks after the course began. He was responsible for conceiving of, and designing, the project description for the e-commerce website. Once the course was underway, he also served as a customer, coach, and mentor 24-hours a day and seven days a week for the duration of the course. This was a big responsibility since the teams were in four time zones and required scheduled and unscheduled support on the edges of the normal work days. The visiting faculty member also had a full-time day job, in addition to serving as customer, coach, and mentor. He provided teams with user stories and acceptance criteria, attended regularly scheduled meetings with each of the three teams, and provided ad hoc meeting support with lead developers. He was also responsible for participating in administrative meetings with UMUC’s faculty and staff, maintaining a delicate balance of institutional versus individual student needs, and providing written feedback on team performance at the end of all iterations, as well as the final product. All-in-all, the visiting faculty member worked full-time on adapting agile methods to UMUC’s capstone course from July 2008 until late December 2008. He was also responsible for preparing the final experience report and getting it published.

A. Team Awesomesauce

The team named Awesomesauce was the first to form, had the most exposure to agile methods, and had the most proficient computer programmers. They chose to use Scrum as their agile method and they also chose an agile project management tool called Version One to help manage their project. They were the last to engage the customer, coach, and mentor and struggled with how to implement Scrum in a virtual environment. Their team, like most of the others, was spread across four time zones. The customer initiated contact with their team, provided them with a broad vision of the project, and gave them a list of initial user stories. Their coach and mentor also contacted them with advice on how to begin their project, manage their iterations, and complete their e-commerce website. The coach and mentor assured their team, like all teams, that he expected them to ultimately succeed and earn their degrees. This little pep talk settled them down and they immediately began work on their first iteration. They had never used Scrum before and they were a little rusty. They decomposed the user stories into development tasks and populated their product and sprint backlogs with low-level technical activities. In other words, the user stories seemed to disappear and prioritization was unclear. The coach and mentor asked them to revive their user stories and maintain both of the lists, the customer’s user stories, their priority order, and the low-level development tasks necessary to implement them. Their first iteration was the most immature of the three teams. However, they nearly completed the entire e-commerce website by the end of the second iteration. They implemented the most user stories of the three teams by the end of the third iteration.
which the first two teams responded nicely. They were prodded by the customer, coach, and mentor to pick up the pace and implement as many of the user stories as possible, to reinforce the concept of teamwork, cooperation, and selflessness. Team Yellowstone implemented an initial version of their e-commerce website before the first iteration and the coach and mentor had to remind them to implement the user stories requested by the customer, rather than implementing unrequested requirements. The team lead was teaching himself as much about the latest web technologies, and would often involve the customer in low-level technical discussions about software implementation. Team Yellowstone implemented the least number of user stories, in spite of their early promise.

### B. Team Kestrel

The team named Kestrel was the second to form, had some exposure to agile methods, and had the second-most computer programming proficiency. Team Kestrel also used Scrum as their agile method and set up a simple Wiki for agile project management, e.g., tracking user stories. They struggled with Scrum and seemed to reject the notion of using Release Planning instead of Sprint Planning as their primary agile project management framework. They seemed to be having intellectual objections to mixing and matching best practices across the various agile methods. They were also one of the last teams to contact the customer, coach, and mentor. The customer initiated contact with their team, shared his vision of the e-commerce website they were to build, and provided them with a list of user stories. The coach and mentor routinely engaged them to guide them along a path to ultimate project success. All of the teams were encouraged to contact the customer with any questions about the user stories, which they did. Their lead programmer developed wire-frames to validate the user stories before implementation. The customer was contacted to approve the wire-frames before the end of the iterations. The first team often made on-the-fly changes during demonstration dry-runs, instead of using wire-frames. The teams were also encouraged to contact the coach and mentor with any questions they had about agile methods, which they did. The first two teams also used the university’s intranet for collaboration. This was monitored by the coach and mentor and was used to gauge the progress of the first two teams to steer them around various roadblocks, as sort of an invisible hand. Team Kestrel’s first iteration was a little basic, but like the first team, they picked up the pace and implemented a fully-functional e-commerce website by the end of the third iteration. Each of the teams were prodded by the customer, coach, and mentor to pick up the pace and implement as many of the user stories as possible, to which the first two teams responded nicely.

### C. Team Yellowstone

The team named Yellowstone was the third to form, had little exposure to agile methods, and had the least amount of computer programming proficiency. Team Yellowstone had the broadest range of industry experience, with both the least and the most experienced individuals. However, their least experienced member assumed responsibility for leading the team. Deeply influenced by the principles of traditional methods taught to him by UMUC, he offered a lot of initial resistance to the use of agile methods. However, once his initial inhibitions were addressed, he offered the least continuing resistance to the use of agile methods and methodically executed the Release Planning methodology. This team also used a Wiki for agile project management. Unlike the other teams, they were the first to contact the customer and established a close and continuing relationship with the customer, coach, and mentor. The lead developer felt burdened by the computer programming task at-hand, so the coach and mentor guided his team towards the use of WebEx and Skype for successfully implementing virtual pair programming sessions. This eased the burden on the team lead, the rest of the team felt more involved, and they were eventually able to complete their tasks. It’s not unusual for academic teams to struggle with the concept of cooperation, and this team was certainly one of those. This team required a lot of extra coaching and mentoring to reinforce the concept of teamwork, cooperation, and selflessness. Team Yellowstone implemented an initial version of their e-commerce website before the first iteration and the coach and mentor had to remind them to implement the user stories requested by the customer, rather than implementing unrequested requirements. The team lead was teaching himself as much about the latest web technologies, and would often involve the customer in low-level technical discussions about software implementation. Team Yellowstone implemented the least number of user stories, in spite of their early promise.
methods was a critical success factor.

customer who was very knowledgeable with agile
after each iteration due to pointed feedback. A
detail. The teams exhibited dramatic improvement
where requirements must be specified in excruciating
product of their prior training in traditional methods,
they didn't seem to want it. This could have been a by-
gave teams the latitude to express design creativity,
and get the job done had a higher velocity. Teams who
sessions. Teams whose programmers knuckled-down
complained the most about teamwork required more
and receive the product via electronic distribution.
Each of the teams demonstrated this basic capability.
There were other user stories constituting website
adornments. The team with the highest velocity
implemented more lower-priority user stories, or
website adornments.

The teams with the highest programming velocity
had the least amount of customer interaction. Fast
team also used fewer agile methods practices. Teams
with the lowest programming velocity had the highest
amount of customer interaction, but used more agile
methods practices, such as virtual pair programming
sessions. Teams whose programmers knuckled-down
and got the job done had a higher velocity. Teams who
complained the most about teamwork required more
oversight and had lower velocity. Some teams relied
on the customer to specify the user stories in a lot of
detail using wire-frames to validate them, and others
simply did what they wanted. Although the customer
gave teams the latitude to express design creativity,
they didn’t seem to want it. This could have been a by-
product of their prior training in traditional methods,
where requirements must be specified in excruciating
detail. The teams exhibited dramatic improvement
after each iteration due to pointed feedback. A
customer who was very knowledgeable with agile
methods was a critical success factor.

VIII. RETROSPECTIVE

Each of the teams demonstrated a fully
operational e-commerce website. The teams were
encouraged to perform a dry-run of their iteration
demonstrations to head-off any issues before
acceptance testing. All user stories were agreed upon
and prioritized, before, during, and after the iterations.
The teams were encouraged to use WebEx to involve
all stakeholders. Then, demonstrations were rehearsed
to work out the kinks. The overriding user story was
for the teams to demonstrate the execution of an end-
to-end e-commerce website transaction. That is, the
customer had to find a product, add it to a shopping
cart, check it out, purchase the item with a credit card,
and receive the product via electronic distribution.
Each of the teams demonstrated this basic capability.
pros and cons involving website
adornments. The team with the highest velocity
implemented more lower-priority user stories, or
website adornments.

The teams with the highest programming velocity
had the least amount of customer interaction. Fast
teams also used fewer agile methods practices. Teams
with the lowest programming velocity had the highest
amount of customer interaction, but used more agile
methods practices, such as virtual pair programming
sessions. Teams whose programmers knuckled-down
and got the job done had a higher velocity. Teams who
complained the most about teamwork required more
oversight and had lower velocity. Some teams relied
on the customer to specify the user stories in a lot of
detail using wire-frames to validate them, and others
simply did what they wanted. Although the customer
gave teams the latitude to express design creativity,
they didn’t seem to want it. This could have been a by-
product of their prior training in traditional methods,
where requirements must be specified in excruciating
detail. The teams exhibited dramatic improvement
after each iteration due to pointed feedback. A
customer who was very knowledgeable with agile
methods was a critical success factor.

IX. SUMMARY AND RECOMMENDATIONS

The primary purpose of this capstone course was
to introduce the use of agile methods in software
ingineering education. The use of agile methods is a
formidable challenge. It took two years to instill the
principles of traditional methods in our students. We
attempted to instill the principles of agile methods in
our students in only 13 weeks. In many ways we
succeeded in the application and use of agile methods
in such a short period of time. Each of the three teams
learned a lot about agile methods, applied the practices
of Scrum and Release Planning, and implemented
three fully-functional e-commerce websites. All 15
students passed the class and went on to “earn” their
master’s degrees in software engineering from
UMUC. Agile methods require emphasis on both the
software engineering process as well as the software
product. Teams who struck a good balance between
the agile methods process and proficiency in
implementing web technologies performed better
overall. However, our success was not without its
challenges. We could have all benefited from earlier
coursework in agile methods, and more emphasis on
technical proficiency, virtual collaboration, coaching
and mentoring, knowledgeable customers, feedback,
and teamwork, among many other improvements.

• Early coursework. Training and education in
agile methods should be started as early as
possible. Agile methods should be discussed in
introductory software engineering coursework.
They should be taught along side traditional
methods as a viable alternative. Agile methods
should be taught as a core course or as an elective.

• Technical proficiency. The significance of
technical programming proficiency should not be
overlooked. The goal of the capstone course was
for each of the teams to build an operational and
functional software system. Although many of the
teams were steeped in the traditional principles of
software engineering documentation, the teams
with skilled programmers performed better.

• Agile roles and responsibilities. Emphasis
should be placed on transitioning teams from
traditional to agile methods roles and
responsibilities. Teams assume agile methods are
simply a new process for traditional roles, such as
project manager, analyst, designer, coder, tester,
quality assurance, configuration management, etc.
This is okay if a team has one or more strong
programmers. However, agile methods require
more team members to be active programmers
who perform some if not all of these traditional
roles in addition to coding responsibilities.
• **Virtual collaboration.** Unlike traditional universities, non-traditional programs often involve students in different time zones. This presents some technical challenges. Teams within the same time zone were able to successfully use WebEx and Skype to perform virtual pair programming to share the workload. However, this proved difficult with other teams.

• **Coaching and mentoring.** The importance of coaching and mentoring cannot be emphasized enough. Coaches and mentors were required to train the faculty and staff, design the course, provide orientation for the students, and interact with the students throughout the semester. Whether in an academic or industrial setting, the use of coaches and mentors is required.

• **Knowledgeable customer.** Customers should also have training and experience in agile methods. Many developers are just starting to gain knowledge and experience with agile methods. Teams will have a hard time succeeding with agile methods, unless both developers and customers understand the principles of agile methods and how to apply them correctly.

• **Constructive feedback.** Customers, coaches, and mentors must be willing to provide direct feedback to developers. Developers must also be open to feedback. Customers should be firm with their user stories, requirements, prioritization, and acceptance criteria. Coaches and mentors must be bold enough to suggest process improvements and make radical course corrections if necessary.

• **Teamwork.** The lack of teamwork in both academia and industry continues to be a problem. Fierce individualism is a common characteristic of Western culture. This proved to be the case among the faculty, staff, and students. The teams who cooperated with one another the most performed extremely well and the teams with the least cooperation didn’t perform very well at all.

• **Buy-versus-build.** Developers should consider using commercially available web products and services first. Developers should not reinvent the wheel by building products from the ground up using the latest third-generation computer programming languages. Developers can achieve a higher level of productivity (velocity), quality, and customer satisfaction, if they utilize commercially-available web services.

• **Aesthetic design.** Developers should be provided with training and education in the principles of aesthetic design. Poor website quality costs billions of dollars in lost e-commerce revenues each year. Literature is starting to emerge, which attributes poor aesthetic design to agile methods.

**Further Reading**


