Oct 17th, 10:09 AM - 10:27 AM

Incorporation of Agile Development Methodology into a Capstone Software Engineering Project

Jon G. Kuhl
University of Iowa

Incorporation of Agile Development Methodology into a Capstone Software Engineering Project Course

Jon G. Kuhl
Department of Electrical and Computer Engineering,
The University of Iowa,
Iowa City, Iowa

Abstract

This paper describes the author's experience in transitioning an undergraduate capstone software engineering project course from a traditional "waterfall" format to a modern agile development methodology. The agile approach replaces the sequential, and documentation-intensive, product development steps of the waterfall model--requirements analysis, system design, implementation, testing, and deployment/maintenance--with a series of short development sprints during which team members collaboratively develop and release a small set of new product features. Agile development focuses on collaborative teamwork and effective communication rather than a strictly managed and rigorously documented process and stresses continuous testing, integration and deployment rather than deferring these steps until late in the development cycle. While agile methodologies have gained considerable traction in industry, their adoption by academia has been slow due to several factors including challenges in effectively monitoring and mentoring student teams, difficulties in carrying out formative and summative evaluation of student work, and academic bias against a process sometimes regarded as non-rigorous. The author attempted to address these challenges in spring, 2014 while redesigning a long-standing capstone SE course to an agile methodology.

Background

For nearly two decades, the author has been teaching a semester-long, team-based software engineering project course. This course serves as the capstone for a sequence of software engineering courses taken as electives by upper-level undergraduate students in Electrical Engineering and Computer Science at The University of Iowa. Students in the capstone course have completed at least two prior software engineering methodology courses or gained equivalent real-world experience. Student teams, comprised of four or five members, are permitted to choose their own projects, subject to instructor approval, and teams are strongly encouraged to seek out real customers for their projects. Prior to the spring, 2014 semester, the capstone course utilized a software development process based on the traditional *waterfall model.*[1] This model derives its name from the fact that it views the software development lifecycle as a one-way through a series of sequential steps as shown below in Figure 1. The inherent "downhill" flow of the model means that mistakes or omissions in early stages of the process may be difficult and costly to correct in later stages. This necessitates a heavy emphasis on front-end planning and documentation. For this reason, waterfall-like software engineering processes are sometimes referred to as *plan-driven development.*[8]

The waterfall-based capstone course heavily stressed front-end requirements analysis and high-level design. At each step, student teams were required to produce extensive documentation, typically including the following documents: project plan, requirements specification, system

architecture document, detailed design document, test plan, deployment plan, user manual. These documents provided the primary basis for normative and summative evaluation of student performance.

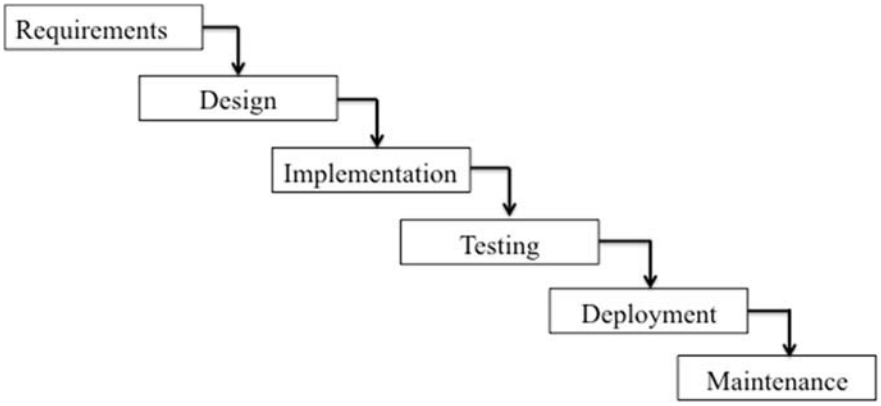

Figure 1: A waterfall lifecycle model

The author noted several ongoing issues with the course, namely:

1. Many of the student teams struggled mightily with conducting and documenting a detailed requirements analysis for their project. Although it provided students with valuable appreciation for the difficulty and complexity of software requirements analysis, the process took extensive time (at least two weeks of effort, sometimes much more) during the semester and often yielded poor results that provided an insufficient basis for downstream development activities. Forcing teams with unsatisfactory requirements specifications to redo the requirements analysis step took still more time from the semester and often yielded only a marginally better result. The inability of customers to clearly articulate their needs contributed to the teams' difficulties in carrying out a rigorous up-front requirements analysis. Often, faced with the prospect that the requirements analysis process would not converge to a satisfactory conclusion, the instructor had no choice but to allow teams to proceed without a set of complete and consistent requirements in place.

2. The amount of documentation required during the course of the semester--at least five major documents--was debilitating and students complained that they were spending more time writing documentation than working on actual software development activities. Many teams coped with the high overhead of documentation by delegating documentation responsibilities to weaker team members, while stronger members focused on design and coding activities. This marginalized some team members and sometimes led to production of documentation that was not reflective of the development activities being carried out by other team members.

3. The large amount of front-end planning, analysis, and high-level design activity during the initial portion of the semester significantly compressed the time available for project implementation and coding, leading to rushed and stressful development in the latter weeks of the semester. In this environment, it was difficult to convince students to pay appropriate attention to comprehensive testing of their projects. In particular, the

compressed timeframe made it unreasonable to require students to master and utilize automated testing tools in their development process. The imposition of dictated milestones for completion of lifecycle steps did not alleviate this problem since the formative feedback provided by the instructor often required teams to rework the results of a lifecycle step before proceeding to work on the next milestone.

4. The student teams lacked the management hierarchy to effectively execute a highly managed development process. Although, team coordinators were designated for all teams, they could not really be expected to function as managers exercising authority and control over other team members. As a result, reaching team consensus on priorities, allocation of work, and other important decisions often proved difficult.

While the capstone course provided a good, if sometimes painful, introduction to the realities of real-world software development, the net effect of the issues noted above was that many teams had a frustrating capstone experience culminating in a disappointing final product. While strong teams were able to perform well, weaker teams tended to bifurcate into coders and documenters working somewhat independently of each other. The constant stream of documentation milestones negatively impacted the morale of some teams. An almost universal experience was excessive stress and resulting inattention to quality assurance in the latter stages of the semester.

The Case for Agile

Agile software development methodologies[2] began to appear in the mid-1990s as an alternative to traditional waterfall-based processes. In 2001, a group of prominent agile proponents authored the well-known *Manifesto for Agile Software Development*[3] that lays out twelve basic principles of agile development. The manifesto addresses a number of issues experienced in the author's capstone course, as described above. These include:

- A focus on team interactions and customer collaborations over formal processes
- Emphasis on development of working software on a short time-scale
- Acknowledgement of uncertainty and fluidity of requirements as a fact-of-life
- Self-organizing teams
- Focus on working software over comprehensive documentation

The most widely adopted agile methodology is Scrum[4]. As shown in Figure 2, Scrum is based on a series of short, fixed duration development *sprints* (typically two weeks in length) carried out by a self-organizing team. The team has at least two designated members with special roles: i) the Product Owner, who represents the interests of the customer and ii) the Scrum Master who is responsible for isolating the development team from outside distractions. Neither the Product Owner nor the Scrum Master is responsible for managing the team. The objective of each sprint is to implement a small set of product features, selected by the Product Owner, and fully integrate them into the product baseline. Features are specified via short, informal user stories, rather than detailed requirements specifications and the set of pending features, known as the product backlog, is assumed to be fluid and flexible.

Each sprint begins with a planning meeting and ends with a review and retrospective to assess the success of the sprint and identify areas for improvement in future sprints. Every day, the team holds a short, focused *scrum meeting* to assess what has been accomplished since the last meeting, identify what will be done by the next meeting and highlight any obstacles or impediments that are impeding progress. Since the intent is to fully integrate new features into a releasable product iteration by the end of each sprint, there is a heavy emphasis on testing, continuous integration, and development of useful (user-oriented) documentation.

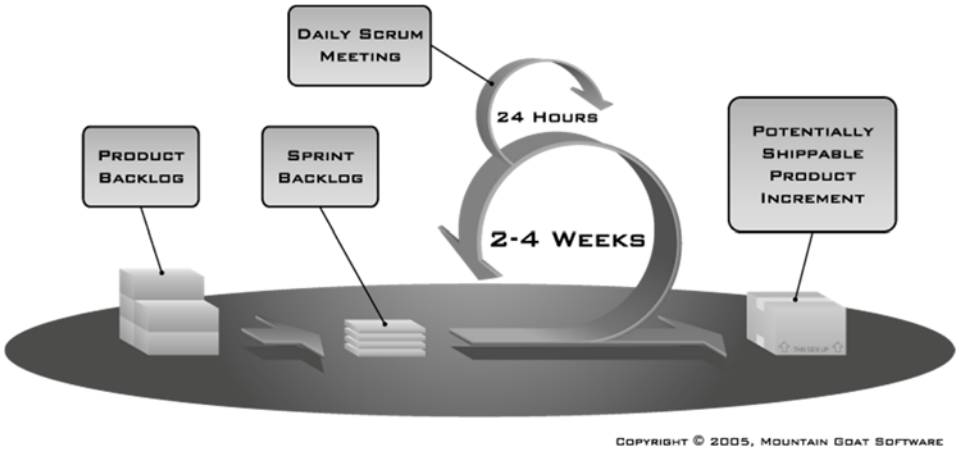

Figure 2: A Depiction of the Scrum Agile Development Cycle [10]

In the past decade, agile development has begun to see widespread industrial acceptance. A 2009 survey found that 35% of Information Technology (IT) companies were using agile teams for at least some of their software development projects[5]. A more recent survey, conducted in 2014, found that 75% of surveyed companies were using agile development for some projects and nearly 50% were using agile development practices for the majority of their software projects[6]. The author's personal experience with recent program graduates indicates that even traditionally conservative companies, such as health care information system providers, are moving toward agile development practices.

Despite industrial adoption of agile methodologies, their acceptance and incorporation into academic curricula has been limited. Most popular software engineering texts[1,7] have been updated to include some coverage of agile methodologies but are still organized around traditional waterfall methodology. The same is true of the IEEE Software Body of Knowledge (SWEBOK) Guide[8]. The 2013 ACM/IEEE standard for software engineering[9] has been updated to address agile processes but only one exemplar course syllabus included in the standard specifically mentions agile development among its list of topics.

Bringing Agile into the Capstone Course

Switching the the capstone project course to agile (Scrum) development presented several significant challenges:

1. How to quickly, but effectively, train student teams to utilize Scrum in order to get teams functioning as early as possible in the semester.

2. How to fill the Product Owner and Scrum Master roles on student teams.
3. How to track team progress.
4. How to identify problems in teams and provide formative feedback.
5. How to insure that teams are employing rigorous testing practices throughout the development cycle.
6. How to evaluate individual student performance for grading purposes.

The problem of getting students up to speed with agile/Scrum was addressed by revising a prerequisite software engineering methodology course to include coverage of agile development. A short (six week) project in the prerequisite course was also modified to use Scrum in order to provide students with some practical experience prior to the capstone course. At the beginning of the capstone course, four lectures were devoted to reviewing agile/Scrum development and other important concepts including testing and continuous integration. These lectures overlapped with the formation of student teams and development of product visions and initial product backlogs by teams so they did not substantively delay the start of project activity. Teams were able to begin their first development sprint at the end of the second week in the semester, allowing for the completion of six two-week sprints prior to the final week of the 14-week semester. The last week of the semester was reserved for class presentations and public poster sessions.

Filling the Product Owner role on the student teams posed perhaps the biggest challenge. Since the Product Owner is responsible for development user stories, prioritizing features, and selecting the set of features to be implemented in each sprint, the role is critical to team success. Teams were strongly encouraged to identify a real customer for their project, and approximately half of the teams were able to do so. However, it was not practical for any of the customers to commit the time or effort required for the teams' product owner roles. Other teams chose self-defined project ideas and thus did not have an external customer. In both cases, teams were directed by the instructor to select one team member to serve as the Product Owner. In cases where the team had a real, external customer, this team member was expected to work with that customer as closely as possible to develop and prioritize user stories, select sprint backlogs, and make final feature acceptance decisions at the conclusion of each sprint. Most of the teams that lacked a real customer found it most productive to do product management activities consortially, with the designated Product Owner playing a limited role. In one case, the team delegated full product management responsibility to the designated Product Owner. Both scenarios seemed to work satisfactorily. However, the instructor found it necessary to watch the teams closely to make sure that product management responsibilities--particularly acceptance of completed features--were being adequately addressed.

The Scrum Master role was found to be less critical. The primary responsibility of the Scrum Master is to insulate the team from outside distractions and interface with management and other stakeholders on behalf of the team. In an academic setting, external influences are minimal and most teams found that they could function fine without a designated Scrum Master. Teams were given the option of selecting a Scrum Master and a few found the role useful for facilitating logistical issues such as arranging team meetings.

Tracking of team progress was facilitated by the use of a project management tool called Pivotal Tracker. Tracker is widely used by agile teams to manage product and sprint backlog and monitor

product development progress. The tool provides a web-based dashboard that makes it easy to monitor the creation and management of user stories and assess team progress in implementing new features. Although the primary motivation for mandating the use of tracker by teams was to facilitate project management, the tool provided an excellent window for the instructor to view how effectively teams were functioning. Via this tool, the instructor was able to identify several emerging team problems during the initial sprint and provide formative feedback to the affected teams to address the identified issues.

An additional mechanism used to assess team performance and progress was in-class review/retrospective sessions. At the conclusion of each sprint, teams were required to provide a brief summary of their post-sprint review and retrospective meetings to the class and instructor. To insure that all aspects of team performance were addressed, the instructor identified a particular area of emphasis for each in-class review--e.g. testing, feature acceptance, configuration management, etc. Following each of these reviews the instructor provided detailed formative feedback to the teams, as needed, to address any identified problems or concerns. Following the third sprint, teams were required to give an in-class demonstration of their product.

Since continuous testing is central to agile development, the capstone course provided a heavy emphasis on all aspects of testing: unit testing, integration testing, regression testing, and acceptance testing. By the end of the third week of the semester, teams were required to submit a Test Plan addressing all of the above-listed testing areas. Teams were also required to utilize automated testing tools, of their choosing to expedite the testing process. In addition to submission of the written Test Plan, and in-class review was conducted. Following the fourth sprint, teams were required to conduct an in-class review and demonstration of their testing procedures and processes.

Individual evaluation of student achievement is difficult in any team-based project course. In this course the difficulty was exacerbated by the democratized nature of the teams and the lack of specifically defined roles for team members. The instructor selected a grading criteria based 40% on individual performance and 60% on team performance. The specific criteria were as follows:

Individual performance (40%):	
Peer evaluation (confidential rating by teammates):	20%
Comprehensive personal log, documenting day-to-day activities:	10%
Subjective Evaluation, based on participation, presentations, etc.	10%
Team Performance (60%):	
Instructor Rating of Final Project Presentation	25%
Instructor Rating of Testing Procedures	15%
Poster Session (rating by three outside judges)	15%
Student Peer Rating of Projects	5%

Discussion

Five project teams completed the first offering of the revised capstone project course. All of the teams successfully produced products that implemented a significant portion of the functionality identified in the initial product vision. In the instructor's subjective opinion, the average quality of the projects improved markedly relative to the old version of the class, with teams

implementing substantially more functionality, and a much-improved level of testing. The customer for one of the teams, an IT services executive, provided the following feedback regarding the performance of his project team:

> *"We are very impressed with the production quality and speed of your project team. Within less than 3 months we have a fully functional web site within an expandable client database. We have been in the software development field since 1994. In our experience this has been one of the fastest software development time lines that we have experienced."*

Four of the five teams did a credible job of functioning as effective agile teams, particularly after the first several sprints. The lack of a true Product Owner did not seem to be a significant deterrent to team success. One team, dominated by some strong personalities, focused mainly on ad-hoc coding and failed to adopt good team-based practices. This team appeared to accomplish little until a "heroic" coding session shortly before the final deadline and showed little evidence of a systematic testing process.

Student course evaluations were uniformly positive with no complaints about excessive workload or the agile development framework. With the exception of the one team mentioned above, end-of-semester stress levels appeared to be much lower than in the past.

The author is encouraged by the outcome of the initial offering and plans to repeat the course using the same format and structure in spring, 2015. One significant change will be to enforce stronger expectations regarding team performance and progress throughout the semester.

Bibliography

1. R. Pressman, *Software Engineering: A Practitioner's Approach, Seventh Edition*, McGraw-Hill, January, 2009.
2. M. Fowler, *The New Methodology*, URL: www.marftinfowler.com/articles/newMethodology.html, Dec. 2005.
3. K. Beck, M. Beedle, et al, *Manifesto for Agile Software Development*, URL: agilemanifesto.org, Dec. 2001.
4. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Prentice Hall, 2001.
5. *Forrester/Dr. Dobb's Global Developer Technographics Survey, Q3, 2009*, Forrester Research, Inc., 2009.
6. S. Ambler, *2014 Agile Adoption Survey Results*, URL: http://www.ambysoft.com/downloads/surveys/AgileAdoption2014.pdf
7. I. Sommerville, *Software Engineering, 9th Edition*, Addison Wesley, 2010.
8. P. Bourque and R. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Zversion 3.0, IEEE Computer Society, 2014. URL: www.swebok.org

9. *Computer Science Curricula 2013, Final Report,* Joint Taskforce on Computing Curricula ACM/IEEE, Dec. 2013.

10. URL: http://www.mountaingoatsoftware.com/agile/scrum/images (Scrum image licensed under a Creative Commons Attribution 2.5 License--may be copied, displayed, and distributed with attribution to Mountain Goat Software.)