AC 2011-1103: AGILE METHODOLOGIES FOR HARDWARE / SOFTWARE TEAMS FOR A CAPSTONE DESIGN COURSE: LESSONS LEARNED

Richard Stansbury, Embry-Riddle Aeronautical Univ., Daytona Beach

Richard S. Stansbury is an assistant professor of computer science and computer engineering at Embry-Riddle Aeronautical University in Daytona Beach, FL. He instructs the capstone senior design course for computer and software engineering. His current research interests include unmanned aircraft, certification issues for unmanned aircraft, mobile robotics, and applied artificial intelligence.

Massood Towhidnejad, Embry-Riddle Aeronautical Univ., Daytona Beach

Massood Towhidnejad is a tenure full professor of software engineering in the department of Electrical, Computer, Software and System Engineering at Embry-Riddle Aeronautical University. His teaching interests include artificial intelligence, autonomous systems, and software engineering with emphasis on software quality assurance and testing. He has been involved in research activities in the areas of software engineering, software quality assurance and testing, autonomous systems, and human factors.

Jayson F Clifford

Jayson Clifford is a Research Associate at Embry-Riddle Aeronautical University. He has worked on a number of projects involving the development of unmanned vehicle systems and software processes for small teams.

Michael P Dop, Embry-Riddle Aeronautical University

Michael Dop is a Graduate Student in the Accelerated 5-year BS/MSE in Software Engineering at Embry-Riddle Aeronautical University. His interests are in unmanned systems.

Agile Methodologies for Hardware / Software Teams for a Capstone Design Course: Lessons Learned

Abstract:

Agile methodologies aid software engineering teams by defining the tools and processes necessary to succeed. These methodologies can be tailored to support engineers from other disciplines such as computer engineering. This paper discusses a four year effort at Embry-Riddle Aeronautical University to integrate agile methodologies into our capstone design course for computer engineering and software engineering. This paper discusses the positive and negative aspects of using agile methodologies, and how the faculty has adapted agile processes/tools to better support larger and more multi-disciplinary teams. It concludes with some lessons learned for faculty considering using agile for their design projects.

Introduction:

Agile methods provide an alternative to more rigorous engineering design processes through iterative design and development. An engineering design process guides a team through the development of a product or products from inception to delivery based on customer needs. Software and systems engineering communities have defined a large number of processes that can be characterized by models such as the waterfall model, v-model, spiral model, etc[10]. These models include large monolithic requirements and design activities with feedback loops. Some implementations such as the Team Software Process[6,7] cause a significant amount of overhead for students from stringent requirements in documentation and data logging. Unfortunately, during a capstone design project in which students are working with new tools, programming languages, multidisciplinary domains, etc., they often encounter setbacks, requirements changes, and design changes because they are learning. More traditional processes do not lend themselves to change.

An agile method is a software engineering, or sometimes systems engineering, process that is designed with higher flexibility with regard to engineering process. Agile was first defined with the Agile Manifesto[1,2], which emphasizes frequent and rapid delivery of working prototypes to the customer that are iteratively developed and refined. The teams define the specifics of their own design process based on the guidance of the agile methodology selected. It is anticipated that it will be revised periodically based on the team's experience on the project. Since the team designates the process (with some input from the team's manager), the overhead required can be significantly reduced. There are a variety of agile methodologies that exist within the software engineering community such as Crystal Clear[3], Extreme Programming[4], Scrum[4], etc.

At Embry-Riddle Aeronautical University (ERAU), the Crystal Clear[3] agile methodology has been utilized with some adaptations for the past four years as part of its capstone senior design course for computer and software engineering students. Over that time, the course makeup has shifted from being over 75% software engineering students to only 50%. As a result, these methodologies, which were developed for software teams have been adapted to accommodate both hardware and software design elements. The size of the team has also grown resulting in additional lessons learned regarding the scalability of the process. As emphasized by Cockburn[4],

agile methodologies only define guidelines on the process carried out by the team, but it is up to the team to adapt to what best suits their organization.

Since agile processes such as Crystal Clear are designed for small (below 10 members) software development team, the authors adjusted it to address the changing demographic of the class participants. This paper discusses the implementation of an agile process for a relatively large team with students majoring in computer engineering and software engineering. This paper will first introduce the Crystal Clear agile process. This will be followed by the discussion of the modified process. Finally, the advantages and disadvantages of modified process are discussed. We hope this paper serves as a guideline for course instructors who are considering going agile for a capstone design course for computer engineers, software engineers, or multi-disciplinary teams.

Crystal Clear Process

Crystal Clear is designed specifically to work with small to medium sized teams. Some of the properties of this process include: frequent delivery via 2–4 week iterations; process improvement via reflection workshops at the end of each iteration; osmotic communication by co-locating teams, and utilizing charts and boards to share information; personal safety; focus through a flexible plan that identifies fixed deliverables per iteration; and a technical environment capable of supporting automated testing, configuration management, and frequent integration of team software. The following is a high-level description of the above mentioned properties.

The two primary goals of the process are to: deliver usable/tested software and teach students how to work in teams on highly challenging projects. This process aims to accomplish that goal by: instilling a sense of personal safety and accountability, breaking the project into very small easily implemented parts, and being highly adaptive. To facilitate, there are several tools and activities that can be used throughout the process such as stand-up meetings, walking skeleton, early victory, iterative development, reflection workshop, essential interaction design, project miniature, and bazaar planning.

Stand-up Meetings

Standup meetings are common across the majority of agile processes encountered. Each work day, before work begins, a standup meetings occurs; first, between the members of each individual team leads followed by a standup meeting between the team leads and faculty mentors (if present).

A standup meeting should involve the participants standing. They are more prone to focus and keeps them away from computers and other distractions. A standup meeting should last between 5 to 10 minutes. Its primary goal is to boost accountability and awareness between the team members. Students are asked to face their peers and honestly discuss the progress made. This should be conducted under some level of personal safety such that participants should be willing to admit their development issues without undue ridicule or retribution.

Walking Skeleton and Early Victory

The walking skeleton is an essential element of Cockburn's Crystal Clear process[3]. A walking skeleton is a functional system of hardware and software representing the work to date. With each iteration, the teams add more features to the skeleton, fleshing it out, toward the final whole. Toward the end of the project, the walking skeleton can be refactored into the final deliverable. The design of the current skeleton is usually communicated through diagrams such as those shown in Figure 1.

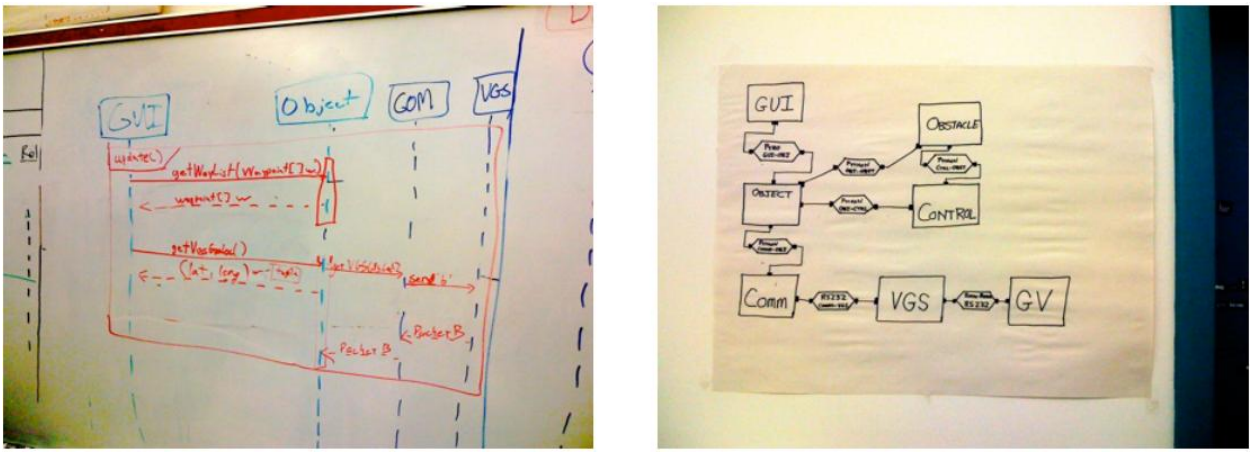

Figure 1: Artifacts from osmotic communication of the walking skeleton's design.

Unfortunately, given the immaturity of students, they are not prepared for complex project planning, and they have had a mindset of ad hoc development. As a result, if dependencies between teams are not adequately identified, the students will often choose to work out completing their features followed by hacking the feature into the skeleton, if possible. On occasion, this results in "trailing limbs"; i.e. features that are implemented, but not well attached (i.e. integrated). When features are not integrated, they cannot be adequately tested, and there is the risk that design could change elsewhere such that they must later be discarded.

Iterative Development

The project timeline is divided into small segments (or iterations) with well-defined deliverables. At the end of each iteration, the deliverables are examined by the customer or a designated representative user to provide feedback regarding the product to date. These iterations allow the teams to be highly adaptive by decreasing the length of feedback loops between customer and developer.

For this course, it has been determined that each iteration should last no more than two weeks. At two weeks, students have enough time to fully realize new features and integrate them into the walking skeleton, but on a short enough time scale that they can remain constantly busy and focused.

Blitz Planning (Project Planning)

Blitz Planning is based on the Extreme Programming planning game[3,4]. Students write out features to be delivered from the final product on a note card (as shown in Figure 2). On the front side of the note card they write out the task to which it is assigned and the number of iterations required for completion. On the back, a detailed description of the task is provided. The note cards are then organized chronologically for each team. Next, the teams work together building a project timeline and identifying issues where the critical path dictates task dependencies. The result is a set of high-level tasks that must be completed by the end of each iteration.

Figure 2: Task card layout for Blitz Planning

Bazaar Planning (Iteration Planning)

Motivation and fairness of work is a major issue. During the 2009-2010 academic year, hard working students often complained that they had to take up the slack of their peers and there was not enough accountability for those that did not complete their work. Work was assigned to the team, but they really worked ad hoc in dividing up the later.

Bazaar planning is a simple concept and easy to implement. From the index cards generated in the blitz plan, a set of tasks small enough for one person to complete in one iteration are generated. These tasks are written on pieces of scrap paper and put into a hat. Students draw from the hat in a random order. With each student, after drawing a task, they can either accept the task or swap tasks with anyone that has already drawn. This continues until everyone has a task. If two students agree, they can choose to swap their task. Once this process is complete, the obligations for delivery are set for the remainder of the iteration. This is similar to the "white elephant" game that is played at gift exchanges near the winter holidays.

Reflection Workshop

The reflection workshop is another part of Cockburn's Crystal methodologies[3]. It is also included in some form for the majority of agile methodologies. It is the opportunity for the students to address the issues that exist with the process and/or the technical environment. Students identify what their current problems are, identify possible solutions to these problems, and indicate which past solutions have worked and they wish to keep. The meeting should occur at the end of each iteration and last for no more than 30 minutes. The results of the workshop should be written down and posted so that all can see and reflect back upon the lessons learned.

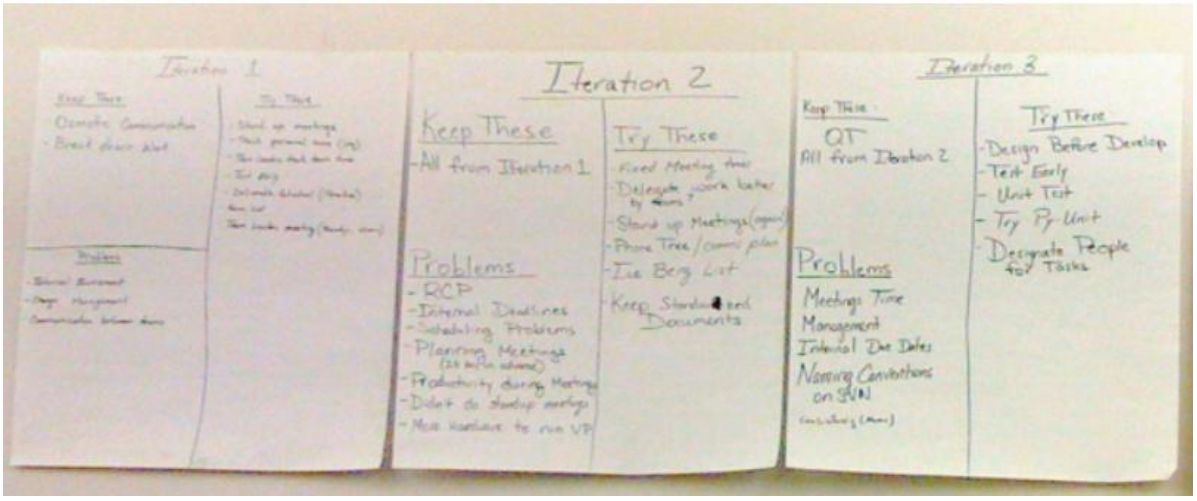

Figure 3: Reflection workshop posters from four iterations.

Test-Driven Design

For each iteration, the students are assigned tasks that they must complete. Test driven design has been added to provide some additional formality to encourage students to frequently test and integrate new hardware and software. While this is an activity developed for programming, it has been adapted for hardware, as well, and is meant to be one of the mechanisms to bridge the gap between computer engineers and the use of this design process.

As a feature is added, before it is implemented, the test is designed. For software, this can either be a unit test, or it can be a test script. For hardware, this will be a test script identifying the inputs, expected outputs, and the measurement technique. In both cases, the test is run before implementation to ensure that a failure occurs. Next, the feature is implemented such that it will pass the test. Once the feature is implemented, the test is re-ran. If it passes, the software is refactored or hardware is better packaged, and integrated into the system. Then, regression testing is performed to ensure that all past tests are also passing. For hardware, this may be a bit onerous, and is still being tuned to be more efficient. For software, automated testing tools simplify regression testing.

Other Tools

Osmotic Communication: As was shown in Figure 1, providing an open channel for communication is essential for teams utilizing agile methodologies. Osmotic communication means that data is presented openly throughout the room such that team members can by "osmosis" absorb the current state of the design, the current state of their team's work, and what should be worked on at any given time. Part of this technique is that the communication never becomes onerous. Our lab utilizes poster paper and the dry erase board, and, typically, there is greater concern for the quality of how the knowledge is transferred that the aesthetics of their

presentation (i.e. rather than using a graphics editor such as Microsoft Visio to create attractive flow charts, they rely upon "quick-and-dirty" sketches to convey their knowledge).

This asynchronous form of communication works well for sub-teams that are co-located. It is much more challenging when sub-teams meet at different times, sub-teams that work at different sites, etc. The use of wikis, message boards, and discussion lists can be used as an alternative, but have been found to be less useful to our students.

UML: The Unified Markup Language (UML) provides a common language that engineers in software engineering and computer engineering can use to communicate the design of software and system activity. Use cases, sequence diagrams, and class diagrams can all be defined clearly and concisely using UML. Software tools exist, and are available to support UML.

Advantages and Disadvantages of the Crystal Process

Some of the advantages of the agile were immediately evident. Through continuous integration into the walking skeleton, the students and instructors could easily visualize the progress being made. While some deliverables could slip in time, often reflection on the process could identify the issues to be addressed. Through osmotic communication via a collocated team, and public display of major design features, the students could share information much more concisely than continuously reviewing a monolithic design documents.

One of the major disadvantages of Crystal Clear was a diminishing quality in documentation. As instructors, a well written design document represents an artifact for both grading and program assessment. Unfortunately, the agile mentality that "busy work" should be avoided often results in less attention being drawn toward documentation. Furthermore, since design occurred iteratively, it was necessary for students to write and re-write documentation, which is somewhat onerous. This concern was discussed in a Software Engineering Institute study regarding agile methodologies[9]. This issue was addressed by periodically evaluating the state of the documents, and then providing feedback to the students.

In order to overcome these disadvantages, we decided to use traditional Waterfall development life cycle during the requirement, design, and analysis. This means students were required to develop the complete system requirements, followed by the Fagan inspection[5] of the requirement. Once the requirement phase was complete, students developed very high-level system architecture. At this point we asked the students to follow the Crystal process. Additionally, we still require the traditional development artifacts (system design specification, test plan, etc.) in the form of living documents. This requires students to develop and update (add/delete/modify) these artifacts throughout the project life cycle. These updates are expected to be done at each iteration.

Past Projects

At the ERAU, agile processes were first introduced during the 2007-2008 academic year, and continued thereafter. In this section, we briefly describe the projects (scope and domain) that were used during each academic year.

Academic Year 2007-2008

There were 12 (8 software, 2 computer engineers, and 2 computer science) students working in a single team that has been divided to four sub-teams during this academic year.

Students were required to develop an autonomous ground vehicle capable of navigating through a search area conducting surveillance activities looking for objects of interest. The vehicle is responsible for avoiding obstacles in its path, using artificial intelligence techniques. The development team was divided to four sub-teams: navigation and control, robotics hardware, communication system, and graphical user interface.

Academic Year 2008-2009

During this academic calendar there were 16 students (10 software, 5 computer engineers, and one human factors) working in a single team, divided to four sub-teams.

The students were required to work as part of a bigger team, about 50 students from five different engineering programs, working on the EcoCAR project. The EcoCAR Challenge[11] is an international competition in which 17 universities compete to produce the next generation of hybrid vehicles over a three year period from 2008 - 2011. This competition is the successor of a number of previous challenges hosted by the United States Department of Energy including Challenge X. All competitors receive a donated vehicle from General Motors as well as electronics hardware, software, tools, and mechanical parts from a number of industry sponsors. Students in the computer and software engineering class were responsible for the development of Intelligent Drive Efficiency Assistant (IDEA)[8]. This intelligent supervisory control system is designed to observe the hybrid drivetrain's control system, the vehicle's state, and the anticipated future state of the vehicle (based on GPS, terrain maps, etc.) and make recommendations regarding which hybrid configuration the vehicle should operate. Unlike traditional hybrid vehicle control systems, this system preemptively selects the "best" hybrid mode so that it is ready ahead of demand whereas traditional control systems react when the current demand on the drive train changes.

Academic Year 2009-2010

During this academic calendar, there were 23 students (15 software, and 8 computer engineers) working in a single team, divided to four sub-teams.

This project was sponsored by Rockwell Collins, Inc. and involved a multi-robot air and ground vehicle that in a collaborative environment conduct search and rescue. Students were divided into four teams: ground control station (image processing, user interfaces, and communications), unmanned aircraft system, and unmanned ground vehicle. The aircraft sent streaming video to the ground control station, software on the ground control station identified potential targets (i.e. bodies of wounded or deceased), a route to the target would be calculated, and the unmanned ground vehicle would drive out to the target to investigate.

There were number of problems throughout this semester that some were due to the team size, some related to the complexity of the project, and some was basically were due to the personnel involved in one of the dysfunctional team. Discussion of these problems is beyond the scope of this paper.

An Adaptation of Crystal Agile Methodologies for Capstone Design Course

After the issues that arose from the 2009-2010 academic year, a significant reflection by the instructors and students necessarily occurred. A student from the dysfunctional team and another student that had graduated with a B.S. and M.S. from our program (and the initial student proponent of agile methodologies for student projects) spent several weeks analyzing the team, the process, and how the process was presented to the students. They presented a report to the course instructors with their suggestions. With some minor modifications, these suggestions were used to adapt for 2010-2011 academic year.

The overall course structure remains quite similar with the addition of three project miniatures (described below). Once those projects are completed, and some team building has occurred, the students perform a requirements elicitation, generate and inspect a requirements document, and perform a high-level design of the final product.

Once completed, the following methodologies were utilized to support the team and its processes.

Project Miniature

There are a few concepts that may be foreign to students entering the senior capstone course including nebulous project definitions (by design), longer time scales than past academic projects, complex problem solving beyond their current academic experiences, and working with a large team.

Project Miniature combines Cockburn's ideas of Process Miniature and Technology Spikes[3]. Specifically, it aims to ease students into their new working environment and process, allowing them to use the process and work with new technologies via a well-defined mini-project. It invites the opportunity for them to learn with low risk and less stress as it is not going to be part of their final project. The mini-projects are executed over two to three iterations. Each iteration is comprised of a micro-project in which some functionality is added with the ultimate goal after all iterations being a completed system.

In addition to incorporating project miniatures, we also made some modification to the way we conduct the class. For example, the faculty became more involved during the standup meetings ensuring that the discussion is on track. They walk through the room listening in and occasionally contributing to the ongoing conversation. They are now also observing body language, tone of voice, etc. Faculty will also participate in the team lead meeting before each work episode in order to learn the status of the project from the students' perspective as well as facilitate communication across the teams.

Finally, for computer engineering students, having small and complete deliverables after each iteration helped them realize that iterative design and development of computer hardware is just as feasible it is with software so long as there is a sufficient system-level design of the overall system. These students came aware of development at the module and component level, and the

ability to use bench equipment and other techniques to perform a type of unit testing on their hardware.

Other Changes

In addition to the project miniatures, the other major change was that the faculty instructors played a greater role as process coach (which is a concept in a number of agile methodologies[4,7]). This role is primarily utilized during the reflection workshop period in which students need the most coaching on iterative process improvement. At this time, special care is made to ensure that non-software engineering students adapt the concepts of Cockburn's Crystal methodologies[3,4] to the appropriate analogue for hardware and mechanical problems. Lastly, it is important that team leads are emboldened to take on true leadership roles and ensure that more hardware oriented teams have computer engineering leads.

To improve the students' writing, a rubric has been provided and accompanies each artifact's outline. These rubrics specify the expectations for poor, good, and excellent delivery of the content of the proposals. In addition to clearly identifying the expectations of each section of the paper, the rubrics also define expectations for grammar, spelling, formatting, etc.

Modified Process Execution and Lessons Learned

There are 18 students (10 computer and 8 software engineering) enrolled in the 2010-2011 capstone design course. This section will qualitatively analyze the results to date from this academic year. Additional results will be discussed during the oral presentation at the 2011 ASEE Annual Conference.

Project miniatures:

The project miniatures provided a greater value added to the course than expected for both the students and instructors. The class was divided into four teams with a mixture of computer engineering and software engineering students on each team. Two mini projects were completed that addressed challenges from previous academic years. The projects were designed such that they could be easily divided into a set of tasks that can be shared between the teammates.

The first mini project required the students to create a GPS NMEA 0183 parser that would read NMEA strings from an input stream, parse each received string, and display the results in a graphical user interface. Prior to starting the project, the students were given high-level requirements. The teams planned out their design, assigned deliverables to each member, executed the deliverables, and generated a design document. This was accomplished in a week and a half. Only one team failed to deliver a working product because of poor team planning. A reflection workshop was performed for each team and each attempted to identify their process's strengths and weaknesses.

One major benefit from the first mini project was the ability to identify team leads. With the exception of the failing team, leaders naturally emerged from each of the other teams. This allowed the instructors to better define teams for the capstone project (as discussed later).

The second project miniature was a two week project. It was a larger project involving a variety of new tools. A microcontroller was programmed to receive input a data link. The microcontroller software parsed the incoming data packet and did one of the following: commanded a servo to rotate to a user specified angle (by sending a pulse-width modulated signal), or enable/disable an LED. The students produced a small desktop application for Windows (using any language of their choice) to allow users flip the state of the microcontroller's LED using a toggle button, or using a slider to select the rotational angle of the servo.

Despite the added work load, the students learned from the first mini project, and adapted their process quite successfully to ensure greater success. Unfortunately, the team that had difficulties with the first mini project continued to have leadership and work breakdown issues. They were, however, much more successful that the first mini project.

One major success with the mini-projects was the new understanding that the students had of test driven design. Computer engineers were forced to demonstrate their understanding of bench tools such as multimeters and oscilloscopes. Software engineers created test cases after their initial design, which resulted in more test execution during development.

Capstone Project:

The 2010-2011 project was funded by Rockwell Collins, Inc. The students were challenged to implement a system to detect, identify, and deter wildlife from airport properties. The students must use computer vision to detect and then classify potential intruders on the airport surface. The target's geographic location is then identified given its pixel location(s) within the image, the camera's position, and the camera's orientation. An unmanned ground vehicle (UGV) receives a path to travel from its current location to the intruder. The UGV follows the path avoiding any encountered obstacles, and upon arrival at the target location activates a deterrence mechanism such as flashing lights, loud noises, and/or patterns of robot motion to scare away the wildlife, if present.

The students were divided into three teams. The three successful teams from the mini projects remained intact while the students from the fourth team were distributed into the other teams. This provided a greater work force for each of the three sub-tasks. The UGV team focused upon the development of the unmanned ground vehicle. The deterrence system team developed deterrence mechanisms for the UGV, and developed path planning algorithms to direct the UGVs to their targets while avoiding runway incursions. The third team focused exclusively on the computer vision challenges.

Immediately, the students showed greater interest in the process and using it to achieve success. After the first iteration, a very thorough reflection workshop was conducted in which students were highly critical of what they had accomplished, what were their hindrances in both the technical environment and the process, and a path toward success. For instance, after the first iteration, when the teams saw numerous duplications of work, the teams resolved to include posted task lists to indicate what the team was working on for that iteration, who on that team was responsible for its delivery, and the current state of progress in delivering it (e.g. planning, design, implementation, or testing).

Another major success was the greater level of system integration into the walking skeleton. First, the mini projects reinforced the process's desire for frequent integration. Second, students and instructors implemented a dedicated testbed for system integration. This dedicated set of workbenches, bench equipment, and computers were used exclusively to perform integration testing. Having a dedicated work space for integration encouraged the each team's integration lead to work toward constant integration. Compared to past academic years, there were fewer issues with product integration, and the customer demonstrations often went much smoother.

The student documentation has improved by using the rubrics described above, but there are some concerns. The rubrics forced the students to pay attention to the documentation requirements, to start early, and to actually write the document to meet the expectations of the instructor/customer. Unfortunately, this did backfire as the number of students involved with documentation, and the hours that were put into the documentation, were larger than desired. Students were obsessed with achieving "excellent" under all categories when in many cases "good" would have been sufficient to meet the needs of both the customer and the instructors. To address this concern, the instructors provided a special lesson on task prioritization.

As stated above, at the time of this paper's submission, the students are in the middle of the spring 2011 term. It is anticipated that as the final deadline approaches the process will become more strained and more lessons learned will emerge. These lessons will be shared during the technical presentation at the 2011 ASEE Annual conference.

Conclusion

From our experiences, agile methods are viable for capstone design courses that include students from multiple disciplines. Over a four year progression, these "software" methodologies have been used more readily with computer engineering students. During the 2011-2012 academic year, electrical engineering students will become part of this capstone design course, and their needs likewise incorporated into the process. Some of the tools and techniques used by electrical engineers will also likely be brought into the process.

It was learned that some methodologies such as Crystal Clear do not scale well as they were originally written. If the process is too light weight, there is a tendency for its execution to break down. Without sufficient accountability, some students become prone to contributing less than their fair share. As more sub-teams exist, the design also becomes more complex, and many lightweight processes do not define processes for sub-teams to interact under the umbrella of a single project.

The modifications to Crystal Clear are believed to remedy some of these problems. Part of the solution is instructional. The remainder lies in adding enough accountability and coordination between the sub-teams, and disciplines, such that all students are contributing to the goals of the project while always adhering to the principles of the process.

Some important lessons learned worth nothing in conclusion are the following:

- Team morale is a critical aspect to any team project, but with an agile project it is essential. Given much of the effort on maintaining the process lies in the participants, if the student team begins to fall apart, the process can fall apart resulting in ad hoc design.

- Faculty must be vigilant in observing team, and working to ensure that ad hoc design is not being carried out.
- Extra effort must be put in place to ensure that the necessary tools and resources are at the team's disposal such as automated testing tools, working computers, adequate writing surfaces for osmotic communication, etc.
- Process training is important. Mini-projects provide on-the-job training outside of the final project's deliverables, and can lead to greater productivity once the actual design project begins.
- Leadership is important. Faculty must coach their inexperienced team leads.

References:

[1]AgileManifesto.org, "Manifesto for Agile Software Development." Online at: http://agilemanifesto.org/, 2011.

[2]Agile Software Alliance, "Principles behind the Agile Manifesto." Online at: http://agilemanifesto.org/principles.html, 2011.

[3]Cockburn, A. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Upper Saddle River, NJ: Addison-Wesley Professional, 2004

[4]Cockburn, A. *Agile Software Development: The Cooperative Game (2nd Edition)*. 2nd ed. Boston, MA: Addison-Wesley Professional, 2006.

[5]Fagan, M. "Advances in Software Inspections." IEEE Transactions on Software Engineering, Vol. SE-12, No. 17, July 1986

[6]Humphrey, W S. "*The team software process (TSP)."* Technical Report for Software Engineering Institute, no. CMU/SEI-2000-TR-023, November 2000.

[7]Humphrey, W S. *Introduction to the Team Software Process*. Reading, MA: Addison-Wesley, 2000.

[8] King, A., Del Buono, M., Marolf, J., Dop, M., and Stansbury, R. "An Intelligent System for Improving the Efficiency of a PHEV For EcoCAR Challenge." ASME Energy Sustainability Conference 2009, San Francisco, CA, July 2009.

[9]Paulk, Mark. 2002. Agile Methodologies and Process Discipline. *The Journal of Defense Software Engineering*, no. October: 15-18.

[10]Somerville, I. *Software Engineering*. Eighth Edition, London: Addison-Wesley, 2007.

[11]US Department of Energy. "EcoCAR Challenge", Online at: http://www.EcoCARchallenge.org/, 2011.