AC 2011-1786: WORKING TOWARDS THE STUDENT SCRUM - DEVELOPING AGILE ANDROID APPLICATIONS

Thomas Reichlmayr, Rochester Institute of Technology

I am an Associate Professor in the Department of Software Engineering at the Rochester Institute of Technology. Prior to transitioning to my academic career, I worked as a software engineer in the process automation industry in a variety of roles over a span of twenty five years. My teaching and research interests include the development of undergraduate software engineering curriculum, especially at the introductory level. Of primary interest is the study of software development process and its application to course curriculum and student team projects.

Working Towards the Student Scrum - Developing Agile Android Applications

Abstract
Student project teams are an integral part of the software engineering curriculum. This paper reports on the classroom experiences of student teams developing Android applications using Scrum. The course in study is a software engineering undergraduate elective in Agile Software Development which used Android mobile phones donated by Google as the development environment for student teams to learn and practice Scrum. Scrum is an agile project management framework increasingly being adopted in the development of commercial software products. When used in an academic setting it provides the opportunity to introduce and practice project management skills involving planning, estimation, tracking and identifying opportunities for continuous process improvement. As the ideal number of team members on a Scrum project is 5-9 developers, Scrum maps well from a size perspective for the typical student team collaborating on a course or capstone project. While Scrum has specific project roles and ceremonies, it is intentionally non-prescriptive on the development practices to be used in the execution of the project. In a software development project these practices are realized in the familiar software engineering life-cycles activities of requirements-analysis-design-code-test-deploy. In an Agile process these activities occur in more tightly wound incremental and iterative time boxes. Agile has also introduced variations on existing software development practices such as test first design or the frontloading of testing activities early in the development life-cycle. The paper addresses the benefits and limitations of the adoption of Scrum by a student project team and proposes recommendations for a pragmatic process framework – the Student Scrum – based on the contributions of agile processes and practices.

Introduction

Prominent among the outcomes for our undergraduate program in Software Engineering at the Rochester Institute of Technology is:

"By the time of their graduation, all students will have demonstrated the ability to work in small teams to develop software systems. This includes the ability to assume distinct operational roles (e.g., project management, quality assurance) in addition to design and implementation."

It is an outcome that is addressed in every course we offer in our curriculum. The challenge of incorporating student team projects into the classroom is reinforcing those activities that, as stated in the outcome above, are in addition to design and implementation. As with other software engineering course projects, there exist hurdles to introducing software process methods and practices. Examples of process practices include requirements elicitation and specification, estimation, scheduling of activities and releases, project tracking, managing risk, quality assurance, collection and analysis of metrics and effective team mechanics. Common hurdles to introducing software engineering process in student projects are cited in similar experience reports. They include compressed time schedules [1], justifying the relevancy of following a process and simply a general disinterest by students fixated on technology and implementation details [2].

Over the past ten years Agile software development practices have grown in acceptance and have gained a solid foothold in commercial software development. [3] Companies from Yahoo [4] to Lockheed Martin [5] are using Agile processes and practices in the development of their products and services. Software engineering course curriculums and capstone projects have also embraced the inclusion of Agile in the classrooms and lab. [6,7,8,9] When one hears the expression "Agile practices", they typically associate the expression with implementation side development activities – pair-programming, test first design, continuous integration and refactoring. These Agile practices are best capitalized on when they are applied within the context of a defined software process framework which addresses the process practices previously noted. Process frameworks provide the structure and discipline required to perform fundamental project management activities. They complement the how-to side of Agile implementation methods, with the who, what and when process activities required to complete a successful project.

The most widely used Agile process framework is Scrum [10]. Scrum has a well defined approach and framework for organizing and controlling a software development project. It is almost always described in combination with Agile software development practices, but is flexible enough to even be used for managing non-software projects. Selecting Scrum as the framework for student team projects has the advantage of introducing software process at a level of ceremony that captures foundational software engineering practices and is manageable within the constraints of a class or capstone project.

This experience report will provide a brief primmer on Scrum to define common terminology and practices. A description of the Agile Software Development course project using Scrum as the development methodology for Android phone application development follows. The report concludes with the challenges and opportunities when using Scrum for student teams in software engineering courses and capstone projects.

Scrum Background

Scrum is an incremental and iterative process framework that, while typically associated with software development, can be used for managing projects in a variety of domains. Scrum as a software development framework was jointly developed and introduced by Jeff Sutherland and Ken Schwaber [11] in the early 1990's. It was inspired by Hirotaka Takeuchi and Ikujiro Nonaka in a 1986 publication [12] that presented a new approach promoting speed and flexibility in commercial product development based on their research in the automotive, photo and printer industries. Their approached proposed overlapping, cross-functional development phases as opposed to sequential phases. The metaphor used was that sequential phases represented a relay race in which members of the relay team waited for the baton to be passed before progressing. The new approach was equated to a rugby team which continually passes the ball back and forth as it progresses down the field. Based on this rugby analogy, "Scrum" was selected for the name of process frameworks used by a project exhibiting this collaborative behavior.

The flow of a Scrum project is a sequence of time-boxed iterations or *sprints*. Each sprint produces a potentially shippable increment of business value. The duration of a sprint is typically 2-6 weeks. Project stakeholders have the opportunity to use these incremental releases and provide timely feedback that can be incorporated into subsequent sprints. The estimated release date(s) of a project is driven by the team's *velocity*, or rate of working functionality completed per sprint. As not all features are of equal size or effort, features are estimated using a unit-less value, and velocity is captured in terms of this relative ranking. A common practice is the description of product features in *user story* format. User stories are descriptions of functionality written in the format:

> "As a <user role>, I want to <feature>, so that I can <business value>"
>
> For example,
> "As a student, I want to view all available courses, so that I can register for next semester"

A user story is sized in scope so that it can be implemented within one sprint. It also provides the acceptance criteria to validate the feature upon its completion. The explicit details of what the feature will do are documented in the acceptance test criteria from which acceptance test cases are generated. The acceptance test plan then becomes an "executable functional specification" or "specification by example" [13] and is the primary source of "How does this feature work?" It is always up to date since it is continuously evolving and being executed by the current version of software.

Requirements are defined and managed in the *Product Backlog*. Although requirements are commonly maintained as user stories, there are no restrictions for documenting requirements in alternative formats. A common misconception is that Scrum, and Agile projects in general, relieve the development team of creating and maintaining project documentation. As a framework, Scrum is non-prescriptive in the types of artifacts and definition of methods used. It is the responsibility of the project team to identify those artifacts and account for the time and resources to see they are properly supported. There is a period of preparation and planning (*Sprint Zero* or *Iteration Zero*) that is needed at the beginning of the project to create the initial version of the Product Backlog, develop the architecture or high level design and perform any other activities needed to support the start of the first sprint. It is during this period that required project artifacts and their content are identified.

The Product Backlog is prioritized by the order in which features are to be added to a sprint for development. There may be a variety of characteristics influencing the backlog priority – business value, time to market, technical dependencies, etc., but the Product Owner is ultimately responsible for the ordering of the backlog. The completeness of user stories in the Product Backlog reflects their relative priority. Stories that will not be worked on until later sprints (bottom of the backlog) may require more detail, while stories approaching the sprint in which they will be developed (top of the backlog) are more complete in terms of definition and acceptance criteria. As stories are developed they also need to be estimated. This facilitates short term (sprint) planning and longer term (release) planning. It is normal for stories to evolve as they are developed during a sprint. Common examples are the clarification of a user story's definition or acceptance criteria.

During sprint planning user stories are pulled in priority order from the top of the product backlog and the team identifies tasks required to implement each stories. Tasks are estimated (in hours) by the team and allocated to team members until all available resource hours for the sprint have been consumed. Tasks and task hours may fluctuate over the course of the sprint as new work is identified or work is removed from the sprint. Remaining task hours are tracked on a Sprint Burn Down Chart on a daily basis and is a visible indicator of the sprint's progress. At the conclusion of the sprint the team's velocity in story points is calculated by adding up the story points estimates from each fully completed user story. Over time velocity will converge within a consistent range enabling more confident predictions on the timing of releases and their expected content.

Structure of the Course

The Agile Software Development course is an upper division, software engineering process elective in our program. It requires that students have completed a required course in Software Process and Project Management during which they have received an introduction to fundamental software development practices. Students usually have completed a portion of their co-operative education blocks (co-op), which is an advantage since they have had the opportunity to experience implementations (good and bad) of commercial software processes. An increasing number of students co-op with companies that are using Scrum as their primary software development process.

Teams of 6-7 students were selected at the start of the term and project work began shortly after an overview of Scrum was completed. A challenge of any course related project is finding the sweet spot of scope and complexity. There needs to be enough work to engage an entire team, while at the same time working in an environment that does not require an unreasonable startup investment either in the technology or problem domain. In this course the focus was on process practices, so the implementation side of the project needed to be challenging, but not overwhelming. Just prior to the start of the term, Google put out a call for proposals on using Android phones in the classroom, and we were fortunate enough to secure enough phones for each student in the class to use the entire term. Teams used the open source, Java-based development kit -Android Development Tools (ADT), which runs as a plugin in the Eclipse IDE. [14] As our students were well experienced with both Java and Eclipse, the development of an Android app as a course project was a popular choice. We still needed a short period of "Android boot camp" to get teams up to the point where they were proficient enough to jump off to developing their own apps, but teams got through this period quickly.

The next challenge after selecting a course project is, "Who gets to be the customer?" Unless the course is fortunate enough to have an external customer, the instructor typically assumes this responsibility. In Scrum, this role is played by the Product Owner. The Product Owner is the source of user stories and is responsible for defining the priority of implementation and acceptance criteria. We elected not to have all teams develop the same app, but to let the teams collaboratively define their own apps. In doing so the teams spent the initial part of the project writing their own user stories. One member of the team was appointed Product Owner to resolve proposed feature conflicts. This activity allowed everyone to participate in the writing and estimation of user stories. We included the initial development of user stories and Android boot camp as Iteration Zero activities, with the exit criteria from Iteration Zero being that there would be sufficient work defined to support the start of Sprint 1.

The instructor initially assumed the role of *Scrum Master*. The Scrum Master provides guidance and coaching to the team on following the principles of the Scrum framework. They also help the team to remove impediments that may distract them from the tasks scheduled for the current sprint. As the course progressed, and the students gained more experience with the Scrum process, the role of Scrum Master was absorbed by the student team.

The Android apps proposed by the teams included:

- A training app that could be used by runners to track their distance (using GPS) and time during training runs.
- An app that accessed the university's bus system to alert the user when the next scheduled bus was approaching a bus stop.
- A Black Jack card game which tutored the user in casino betting strategies.

As part of the planning activity teams used "Planning Poker", an Agile estimation practice modeled after the Wideband Delphi method to assign story points to each user story. Story points are based on the relative effort to complete as compared to other user stories. At the conclusion of each sprint, the number of story points completed is used to compute the team's velocity which, helps to plan and track future releases.

An example of user stories and story point estimates from the bus scheduling app:

- As a slow walker, I want to know the closest bus stop that will take me to my destination (5 points)
- As a sleepy student, I want to adjust my alarm so that I will be woken up at the optimal time so as not to miss the bus (3 points)
- As an infrequent user, I want to be able to find out when the next bus for a given destination arrives so I know how long to wait at the bus stop (2 points)
- As a frequent user, I want to be able to save a destination and time so I am notified beforehand (8 points)
- As a visitor, I want walking directions from and to a bus stop (8 points)

Prior to the start of the sprint, the team selects which user stories will be implemented. Each story is decomposed into specific development tasks and estimated in hours. After the team has collectively identified and estimated the tasks to fully implement a user story, team members "sign-up" or commit to taking responsibility for the task. This approach to allocating resources is referred to as *commitment driven sprint planning* [14]. The team has a budget of hours for the sprint and can only commit to tasks that fit within that budget. Tasks may be modified, added or deleted during a sprint, but the status of the sprint is tracked on a regular basis by comparing the scheduled number of task hours remaining to the planned end date of the sprint. Scrum tracks this scheduling metric using *a sprint burn-down chart.* The sprint burn-down chart is a valuable visual feedback mechanism to the team not only during the sprint, but at the conclusion of the sprint when a retrospective is held. Teams use this feedback to continually tune their task identification and estimating technique for future sprints.

An example of a bus scheduling user story decomposed into tasks:

User Story: As a frequent user, I want to know where the nearest bus stop is (5 points)

Tasks:

- Display Google Map (3 hours) – George
- Query GPS for current location (1 hour) - Gordon
- Build list of bus stop locations (2 hours) - Aamir
- Display straight line between location and bus stop (3 hours) – Alaina

A user story is considered fully complete when it successfully meets the acceptance criteria established by the Product Owner. An important agile concept is the "definition of done." It requires the team to work with the Product Owner in defining the criteria that ultimately validates a user story by the execution of the acceptance test plan. It also identifies the state of completion that project artifacts need to be in at the end of a sprint – design documentation, user guides, etc. This approach helps to minimize the "95% done" syndrome that haunts many projects unable to achieve escape velocity prior to a release. Teams were required to identify acceptance test cases prior to beginning implementation of a user story. As the sprint proceeded, acceptance test execution was performed as soon as the functionality required to test a user story was available. This required the teams to plan the order of their work so they were continuously producing operational software in small increments.

Although a number of commercial, free and open source Scrum project management tools are available, students initially maintained their projects manually – poster paper and index cards. The central source of project status is the team's Scrum Board which displays user stories and tasks in progress and the current sprint burn-down chart. Although tools can automate much of this process, the manual approach helped the students to focus on the mechanics of Scrum methods without being distracted by glittery tool features. The public posting and display of project status is a key component of Agile methodologies. Alistair Cockburn refers to such postings as "information radiators" [16]. They allow the team to view the status of the project at a quick glance, more effectively than a tool or website.

Teams also participated in daily stand-up meetings and end of the sprint reflections as prescribed by Scrum. Daily stand-ups in Scrum allow each team member to answer three questions:

1. What did I accomplish yesterday (or since the last stand-up)?
2. What am I planning on accomplishing today (or before the next stand-up)?
3. What roadblocks are in my way?

These meeting are meant to be information gathering, not problem solving sessions lasting no more than 10-15 minutes. The team stands to insure the time limitation is not exceeded. It is important to note that stand-ups are not status reporting to a manager (or instructor), but commitments team members are making to each other. In the classroom it was initially difficult to have the students address each other and not the instructor during the standup. As a result of the stand-up, actions may be initiated to address issues or work on scheduled tasks – pairs may form to collaborate on a task, a design review may occur, etc. At the conclusion of a sprint, the team demonstrates the completed user stories and holds a sprint retrospective. During the retrospective the team focuses on process improvements for subsequent sprints by addressing the following Keep-Toss-Try questions.

1. What worked well in the sprint just completed? (Keep)
2. What should we avoid doing in future sprints? (Toss)
3. What could we do differently in future sprints? (Try)

As a result of the retrospective the teams select the highest priority process improvement suggestions and post them on their Scrum Board. This keeps the process improvement goals visible to the team while working on the next sprint. The subsequent retrospective can then review what progress the team made on those goals.

Teams were assessed on their participation and effort in adhering to Scrum practices and the upkeep of their Scrum Board and Product Backlog. End of the sprint presentations were made to the entire class after which teams were critiqued by the instructor and fellow classmates. In addition to the sprint retrospectives, team members submitted confidential peer evaluations to the instructor. We regularly collect peer evaluations on all class projects to be used along with the instructor's observation in determining a potential adjustment to an individual grade. A learning outcome for the course is not only to have the students execute a project using Scrum and Agile practices, but also to critically evaluate the suitability of Agile methods and practices based on the characteristic of the project and development organization. Students addressed this outcome using case study reports and during class discussions. Each student was also responsible for a research report on a contemporary Agile topic.

Working towards the Student Scrum

Can Scrum be effectively used as a software development learning vehicle for student teams in the classroom and on capstone projects? As with any industrial grade methodology, importing Scrum into the classroom or capstone setting presents both challenges and opportunities.

Challenges for the Student Scrum

The context in which Scrum will be introduced or adopted will be driven by the learning outcomes of the course. Is the course an introduction to software engineering process concepts? Is the course primarily focused on a technical aspect (programming language, web application, etc.) where Scrum is being used by course project teams? Is Scrum the methodology selected by a senior capstone team for their project? In all these scenarios the common denominator is time or more precisely, the lack of it. This is a universal issue for academic projects as the period of time students are available for collaboration may be restricted to class or lab time only. A related time issue is the ability to leverage experiences from completed sprints towards improving future performance. Commercial organizations that have transitioned to Scrum identify the ability to continually refine their development process based on past experience as one of the keys to a successful adaptation. These organizations have the benefit of continuity in team personnel over a period of many more sprints than can be afforded to a student team.

Student teams in our class were able to complete three Sprints in addition to the time spent during the Iteration Zero activities. Although each Sprint was three calendar weeks in length, the actual time the teams spent together working was closer to 8-12 hours. Even though abbreviated, this still provided teams with the opportunity to experience the mechanics of a Scrum project. The length of the Sprint was not allowed to be extended as it emphasized the value of measuring the true velocity of a team by the amount of functionality that was completely done, as opposed to time slips for "almost-there" functionality. (Students were able to identify this phenomenon as their Sprint Burndown charts tended to track up, as opposed to down - typically indicating overly optimistic task estimates or missing tasks all together.)

The initial state of the project's requirements is another factor to consider. Classroom projects need to start with a set of established requirements in order to provide time for the students to achieve the course outcomes relative to other parts of the project (design, construct, test, etc.) In our course the development of user stories was a learning outcome, so we traded off less implementation. Interestingly, in our senior capstone projects we allow the student teams to select the development process they will follow. Although many teams gravitate towards an Agile process, most of their project descriptions start with specific, prioritized requirements. Teams are still encouraged to adopt an incremental and iterative process, but it more closely resembles McConnell's Staged Delivery Model [17] where requirements and architecture are well established and functionality is delivered in stages over the life of the project. There are still opportunities to incorporate Agile practices in this model, however the Agile practices for managing continually forming and changing requirements may used less than in a project with more volatile requirements..

Scrum defines three project roles – Product Owner, Scrum Master and Team Member. The role of Team Member actually is a benefit to student projects as it is defines an intentionally flat organizational hierarchy to encourage shared responsibility for product activities and deliverables. The role of Product Owner, even on the best Scrum projects, is often challenging as it requires an individual who can dedicate exclusive time to working with the team in creating, refining and validating Product Backlog requirements. On our capstone project we have sponsors that can dedicate time for initial requirements elicitation and responding to questions, but not to the extent a dedicated Scrum Product Owner would provide. In the classroom, the source of requirements is the project description and responsibility for clarification falls to the instructor. As a result the volatility of the requirements remains low and the development process again more closely resembles the Staged Delivery Model.

The role of Scrum Master can be tricky, as that individual is responsible for providing the coaching and guidance for adhering to Scrum practices. Commercial Scrum projects use *Certified Scrum Masters* [18] that have attended standardized training courses. Student teams most likely will not have that opportunity, so based on the course context this position may be fulfilled by the instructor, or rotated through the group based on the team's level of Scrum expertise.

Opportunities for the Student Scrum

Despite the inability of student teams to match commercial Scrum environments, there is ample opportunity for them to engage in specific Agile practices within the framework of Scrum that support the fundamental software engineering skills they need to learn and practice.

Scrum is supported by a disciplined set of project management practices for the identification, estimation, planning and tracking of activities and deliverables. Students have the opportunity to develop personal as well as team estimation expertise during each sprint planning session. Since sprints are continuously evaluated on a regular basis, the team is required to make on-going adjustments by modifying the content and ordering of tasks. It allows students to define and organize their work in manageable durations. This process also exposes students to topics in risk management and negotiating feature priorities with the customer. Additionally, a Scrum project can be managed entirely without the need for specialized tools or applications beyond simple spreadsheets.

Among Agile's largest contribution to the software development community has been a renewed awareness in the value of testing throughout the entire project life-cycle. Students and industry alike often view testing as an end-of-the project activity, which usually absorbs the schedule delays of the activities preceding it. [19] Establishing a rhythm of test case definition and execution is a key Scrum practice in eliciting, documenting and validating product features. At the unit or component level, unit testing concepts can be reinforced though test first design techniques and the automation of test cases to support regression testing and continuous refactoring. Since development is proceeding in small increments, a failed test typically points to the last completed implementation step as the source of the defect. Automation also supports the seamless collection of testing metrics – tests run, succeeded, failed per sprint or user story to allow teams to use objective data in evaluating the quality product releases.

Scrum provides teams with a consistent forum for communicating progress and identifying issues. Though deceivingly simple in format, good stand-ups work towards building trust and promoting the self-organizing characteristic which is vital to good team chemistry. Stand-ups encourage a more humane and mature work environment by allowing the team to collaboratively help each other in solving individual challenges.

As Scrum is a framework, and is not prescriptive on the types of project artifacts to be created and maintained, it can be tuned to accommodate the requirements of a course or project. The Iteration Zero activities preceding the first sprint can be designed to support the initial definition of targeted artifacts, such as an architecture or design document. As the project proceeds, tasks are identified during sprint planning to account for the time and resources required to support the artifact's evolution.

Conclusions

The use of Scrum or Student Scrums in the classroom provides a consistent framework for both managing projects and introducing good software engineering practices. Student Scrums may need to deviate from traditional Scrum practices in the assignment of roles and time duration of development sprints. Students derive benefit from the use of Agile practices within the Scrum framework that are related to requirements engineering, project planning and tracking, testing and effective team collaboration. The Scrum framework is flexible enough to accommodate a variety of software engineering courses outcomes and support student teams on capstone projects. The use of Scrum is most effective if teams have the opportunity to complete multiple iterations and apply process improvement initiatives as they progress.

The author would like to acknowledge Google for their generous donation of the Android mobile phones used by student teams in the Agile Software Development class.

References

[1] Boetje, J., *"Foundational Actions: Teaching Software Engineering When Time is Tight"*, ITiCSE '06, June 26-28, 2006, Bologna, Italy

[2] Bernstein, L., Klappholz, D., Kellet, C., *"Eliminating Aversion to Software Process in Computer Science Students and Measuring the Results"*, Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET '02)

[3] State of Agile Survey, 2009, VersionOne http://trailridgeconsulting.com/surveys/state-of-agile-development-survey-2009.pdf

[4] Benefield, G, *"Rolling Out Agile in a Large Enterprise"*, Hawaii International Conference on System Sciences, Proceedings of the 41st Annual, Jan 7-10, 2008

[5] Zwicker, M. *"War Stories – Fighter Jets and Agile Development at Lockheed Martin"*, Agile Journal, April, 2007

[6] Kessler, R. and Dykman, N., *"Integrating Traditional and Agile Processes In the Classroom"*, SIGCSE '07, March 7-10, 2007, Covington, KY

[7] Rico, D. and Sayani, H., *"Use of Agile Methods in Software Engineering Education"*, 2009 Agile Conference

[8] Reichlmayr, T., *"An Agile Approach to an Undergraduate Software Engineering Course Project"*, 33rd ASEE/IEEE Frontiers in Education Conference, Nov 5-8, 2003, Boulder, CO

[9] Schneider, J and Vasa, R., *"Agile Experiences in Software Development – Experiences from Student Projects"*, Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06)

[10] Forrester/Dr. Dobbs's Global Developer Technographics Survey, Q3 2009

[11] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Prentice Hall, 2002

[12] Takeuchi, H. and Nonaka, I., *"The New New Product Development Game," Harvard Business Review, January-February*, 1986.

[13] Adzic, G, *Bridging the Communication Gap – Specification by Example and Agile Acceptance Testing*, Neuri Limited, London, UK, 2009

[14] Android Development Tools (ADT) http://developer.android.com/sdk/eclipse-adt.html

[15] Cohn, M., *Succeeding With Agile – Software Development Using Scrum*, Addison Wesley, 2010

[16] Cockburn, A., *Agile Software Development – The Cooperative Game*, 2nd Edition, Addison Wesley, 2007

[17] McConnell, S, *Rapid Development – Taming Wild Software Schedules*, Microsoft Press, 1996

[18] Scrum Alliance - Certified Scrum Master http://www.scrumalliance.org/scrum_certification

[19] Shepard T., Lamb, M. and Kelly, D. *“More Testing Should Be Taught”*, *Communications of the ACM*, June 2001, Vol. 44, No. 6, pp. 103-108