SAND2013-7030C

How We Successfully Adapted Agile for a Research-Heavy Engineering Software Team

Alfred A. Lorber
Comp. Thermal & Fluid Mechanics (1541)
Sandia National Laboratories
Albuquerque, NM
aalorbe@sandia.gov

Kyran D. Mish
Comp. Solid Mechanics & Structural Dynamics (1542)
Sandia National Laboratories
Albuquerque, NM
kdmish@sandia.gov

In our development team at Sandia National Laboratories we have honed our Scrum processes to where we continually deliver high-performance engineering analysis software to our customers. We deliver despite non-ideal circumstances, including development work that can be categorized as exploratory research, regular use of part-time developers, team size that varies widely among Sprints, highly specialized technical skill sets and a broad range of deliverables.

We believe our methodologies can be applied to many research-oriented environments such as those found in government laboratories, academic institutions and corporate research facilities. Our goal is to increase the adoption of Lean/Agile project management in these environments by sharing our experiences with those research-oriented development teams who are considering using Lean/Agile, or have started and are encountering problems.

In this paper we discuss how we create and prioritize our product backlog, write our user stories, calculate our capacity, plan our Sprints, report our results and communicate our progress to customers. By providing guidance and evidence of success in these areas we hope to overcome real and perceived obstacles that may limit the adoption of Lean/Agile techniques in research-oriented development environments.

Keywords-Agile, Scrum, research, scientific programming, engineering analysis software.

I. INTRODUCTION

A. General Background and Context

We are the Scrum Master (SM) and Product Owner (PO) of the Thermal/Fluids Scrum team at Sandia National Laboratories in Albuquerque, New Mexico. Sandia, which is located in Albuquerque and Livermore, CA, has approximately 8000 employees and is one of the three national nuclear weapons laboratories operated by the U.S. Department of Energy. Our Scrum team resides in the Computational Thermal & Fluid Mechanics Department, which is part of the Computational Simulation Group that has a total of seven Scrum teams and approximately 50 developers.

Our team is made up of 22 full and part-time developers and is responsible for the development, documentation, deployment, and support of fluid mechanics and heat transfer software. The team is made up almost entirely of domain experts with PhDs in various disciplines such as Aerospace, Chemical and Mechanical Engineering, and related technical disciplines such as Physics and Mathematics.

Our software is principally written in C++ and our target platforms are massively parallel computers running various flavors of Unix, many of which are found in the TOP500 Supercomputing Sites [1]. Our development is performed primarily on PC hardware running Linux, and as needed on our target platforms for testing and debugging. Our applications do not have a graphical front end.

Our programs solve the mathematical equations that govern the various physical phenomena needed to simulate the aerodynamic and fire environments experienced by hardware in normal operating and accident scenarios, and the thermal/fluid response of the hardware to these environments. Our users are engineering analysts who utilize our software's capabilities to produce numerical simulations that support design and assessment of engineering systems.

Our group began using Scrum early in 2008 starting with three Scrum teams after three developers received SM training. Over the years Scrum has been successful and what was a pilot program restricted to only a handful of software activities has expanded to encompass all development work in our group. All seven of our Scrum teams utilize three-week Sprints that begin in the same week.

B. Initial Challenges

When we began using Scrum we essentially did it by the book, relying on the processes called out in Ken Schwaber's *Agile Project Management with Scrum* [2] and what the SMs had learned in training. During this initial adoption stage, we encountered many problems, e.g., non-delivery of the stories in our sprint backlog and the lack of trust this caused between our stakeholders and the Scrum team. In addition, our standup, review and planning meetings were too long, delivering too little value to those attending, with attendance dropping until these problems were rectified.

To understand our solutions to these problems one must first understand our environment. It is dominated by two factors, a heterogeneous team and science and engineering research-oriented programming needs.

C. The Heterogeneous Team

The ideal Scrum team is what we view as homogeneous, i.e., it is made up of five to nine full-time developers with overlapping skill sets. Cross-functional ability is continually

improved by employing practices such as peer code review, working in team rooms and pair programming. Because of the overlapping skill set, every Sprint the team members can self-organize and divide the labor amongst them after agreeing with the PO on a sprint backlog of user stories during the planning meeting. Ideally, the result is the repeated delivery of a completed sprint backlog at the end of each iteration [3].

Our team is what we view as a heterogeneous team, caused by four factors that we suspect are common in research environments. The first is staffing realities that cause a mix of full- and part-time developers. Second is development work that requires specialization of development skills for which no individual developer spans the spectrum of technical domain expertise. Third is lack of colocation and fourth is targeted funding.

D. Research-Heavy Scientific Programming

We say our software development is research-heavy because it involves our developers taking the following steps before software can be written: First, our developers must work closely with the our engineering analysts users to identify the needed physics for the problem being modeled. Then, equations that correctly model the physics must be formulated. Finally, these equations must be approximated and algorithms identified so that the equations can be solved via software. The difficulty is typically increased because many of the capabilities developed have few, if any, successful precedents as they are inherently exploratory. Thus the work required is often at the leading edge of scientific and engineering expertise, so much so that the work is often reported in peer-reviewed technical journals.

Once these three steps have been undertaken, software development takes on a more traditional role in that the computer code can be written. However, this often has a research aspect due to ever-changing hardware and software components found in supercomputing venues.

II. WHY AGILE MAKES SENSE FOR RESEARCH

When considering the effective management of the research-oriented aspects of our projects, Agile ultimately has proven to be a good fit. The three fundamental components of Agile, i.e., transparency, inspection and adaptation [3], offer clear benefits to research performed in our production environment. When investigating methodologies and implementation techniques, transparency provides the mechanism for inspection, which leads to adaptation. This implies that researchers benefit from fast feedback from both peers and stakeholders, which can be invaluable when trying to formulate a solution for an unsolved problem, or for finding a similar solution that might be utilized to verify the results of the current problem. It also leads to quick realignment of resources based on current results so researchers do not spend undue time investigating dead-end solutions. Importantly, transparency and inspection also help insure that our priorities align with those of our stakeholders, and that misunderstandings and individual biases do not lead to work that is not aligned with the appropriate solutions.

III. THE CHALLENGES IN APPLYING AGILE TO RESEARCH

A. Why it is Difficult

"If we knew what we were doing, it wouldn't be called 'research', would it?" – Albert Einstein [4].

"Scrum is: Lightweight, simple to understand, extremely difficult to master." – Ken Schwaber and Jeff Sutherland [3].

These two quotes embody why we believe applying Agile in general and Scrum in particular to research projects is difficult despite our arguments for why Agile is such a natural fit.

The culture of the research professional in science and engineering is that the act of performing research is a serendipitous endeavor of discovery that cannot be rushed and must be done in an environment where the researcher is free to explore. All avenues affecting the research program should be explored and understood. In addition, research is often not taught as a team effort: in U.S. universities, the research leading to graduate degrees is specifically emphasized as an original individual effort resulting in a student's graduate thesis or dissertation. Thus, the work of the individual is valued, even when that individual is working as part of a larger research team.

In addition, as the well-established phrase "publish or perish" reminds us, researchers are strongly encouraged to publish in peer-reviewed journals and to present at technical conferences throughout their careers. While teamwork is endorsed in such venues, in practice individual recognition in a researcher's body of work is all-too-often the most important criteria for advancement.

Having been educated and mentored in such an environment, it is not surprising that developers who have earned PhDs and are working in the research phase of a project may be apprehensive about completely opening their work up for continuous review, especially when that work is incomplete.

Thus Agile is what we advocate for use in software development to manage these types of technical uncertainties, but Agile is not what is used when pursuing an advanced degree. So realizing the fruits of Agile requires some changes to the individual aspects of research culture in science and engineering, and some modifications of accepted Agile practices that embrace the culture's realities but still stay true to Agile's principles.

B. The Mechanics of Misalignment

Given the culture of the researcher described above, how does this clash with Agile in general and with Scrum in particular? This potential for misalignment occurs in nearly every aspect. First and foremost, the idea of setting a time limit via the fixed-length Sprint goes against the fundamental idea that one cannot rush exploratory research. This goes against the concept of encapsulating a Sprint's worth of work in a user story [5]. If one cannot predict or force results at Sprint's end, how can acceptance criteria be written in such a way that they will be agreeable to the development team and to stakeholders? And finally, the idea of iteratively reporting

incomplete work can become uncomfortable for the developer performing research. This reporting is done each day during the daily standup and once every Sprint during the review. Developers can perceive this as presenting improperly vetted work at these meetings.

IV. WHEN AGILE IS NOT WORKING

The clearest indication we have seen that Agile is not successfully managing a research project is when the team does not deliver what it promised by the end of the Sprint. In the worst cases, all indications during the daily standup are that the story is proceeding successfully but it is revealed just before review meeting that the work was not completed.

In fiscal year 2011, in 16 Sprints we delivered all the user stories we committed to only 1/3 of the time. Unfortunately, this delivery pattern was the norm, and had been since 2008 when we started using Agile techniques in our enterprise.

Non-delivery is unacceptable because it erodes the trust required between the Scrum team and its stakeholders. Without this trust it is difficult to build a successful business relationship where stakeholders continue to seek the team's services. Non-delivery also leads to waste in the PO role. It is difficult to plan for the next Sprint and beyond when it is not known until the end of the Sprint, when it is too late, what is actually going to be delivered.

Non-delivery leads to endless tweaking of practices by the SM in an attempt to "fix" the team. For developers non-delivery leads to frustration and low morale. Eventually, in search of a solution, the SM, PO, and finally management will continually meddle with the team, risking the autonomy Lean/Agile seeks to create.

Another indication is when in the daily standup developers simply do not discuss some sprint user stories. We will discuss a simple way to address this in the next section.

We have also seen it argued that research work should not be placed under Scrum. That is, that various research work is not suitable for Scrum's focus on delivering a potentially releasable product increment each Sprint [3]. Instead, it is suggested that developers should work outside of the Scrum process for a period until they are ready to implement their findings in the code base. We will also consider these issues in the next section.

V. ADDRESSING THE CHALLENGES

Over the six years that we have been using Scrum our team struggled as most others do in our adoption of Agile. While currently our situation is not perfect, and we never expect it to be, we do tackle a large body of work every Sprint and for over a year have consistently delivered by the Sprint's end what we have promised to our stakeholders. Our stakeholders are generally satisfied with these results, even when the technical risk of the work has proven to be higher than originally estimated. In this section we describe many of the challenges we faced and the practices we implemented to overcome them. The discussion is organized based on various meetings and artifacts of our Scrum process. First let us explicitly state the issues we have seen and then categorize them so they can be more easily addressed.

1) Issues

- The larger the team, the longer the meetings and the more difficult to provide value to all participants.
- Average velocity and story-points per story are insufficient to plan a Sprint due to large team size, fluctuating participation and specialization.
- Having developers report on research, especially non-progress, during a Sprint can be difficult due to the institutional culture of research.
- Writing research stories that are concrete can be difficult due to inherent uncertainty.
- Moving research forward at an adequate pace requires transparency, inspection and adaptation.

2) Categories

1. Efficiency: A larger, heterogeneous team amplifies the need for efficient practices.
2. Delivery: The best way to satisfy customers is to simply deliver what is promised every Sprint.
3. Managing Uncertainty: We must manage the uncertainty inherent in research in order to deliver.
4. Part-Time Developers: Variation in team size and Sprint capacity is a reality we must accept.
5. Specialization: Specialization of skills at the PhD level is a reality we must accept and address.
6. Work Within Scrum: It is tempting to perform research related work outside of the Scrum process.
7. Research Visibility: Research benefits form transparency, inspection and adaptation.

B. Addressing the Issues

To begin with, all involved with the Scrum process must understand what is expected of them. We have found that it is extremely effective to explain why these expectations are set in terms of others who are relying on the information. Our expectations include the following:

1) Daily Standup

When we started Scum, our daily standup meeting followed the standard format of a 15-minute meeting in which each developer answered the standard three questions [3]. As our team size grew and with it the number of stories in the backlog, it became difficult to associate what each developer said with specific stories. In addition, it was common for a story in the sprint backlog not to be reported on even when work was occurring, as the developer did not think the work done warranted reporting. This was especially true of research stories where slow progress and dead ends are difficult issues for researchers to report on for the reasons discussed above. We often reached a point where a few stories in the sprint backlog were nearly forgotten because they were not discussed during the Sprint and there was no clear line of responsibility for story completion. As the team size grew, the daily standups tended to start late due to late arrivals and run long due to a lack of focus. This, along with some disarray in reporting, caused developers to stop attending, as they were not finding value in the meetings. Below are the practices we used to address these problems.

Stories: *Story Shepherding*

Practice: Every story in the sprint backlog has a developer who is a shepherd. While not responsible for doing the actual work, the shepherd is responsible for ensuring that a story is completed. This includes ensuring the story gets adequate resources, renegotiating with the PO if needed and that a done slide (discussed later) is created. The shepherd is also responsible for ensuring the story gets reported on during the daily standup even when they cannot attend, typically through an appointed surrogate.

Motivation: In a large, heterogeneous team, it is easy for sprint user stories to not be completed due to lack of discussion, inadequate resources and inattention to detail.

Categories Addressed: Delivery, Research Visibility

Daily Standup: *Task Based Review*

Practice: During the daily standup each story in the sprint backlog is discussed in sequence. Since we want to keep WIP (work in process) low it is understandable that, especially at the beginning of a Sprint, "not started" is a valid status of a story.

Motivation: Given the size of our team and the number of stories in the sprint backlog, stories can be missed or forgotten during the standup. In addition, if multiple team members are working on a story, it is difficult to present the work done on a story in an organized fashion if those members don't report on the story consecutively.

Categories Addressed: Efficiency, Research Visibility

Daily Standup: *Final Chance to Speak*

Practice: At least a minute will be reserved at the end of the daily standup for additional topics to be raised.

Motivation: Given a Task Based Review some developers may not get a chance to speak or bring up related issues, e.g., impediments. This is the time for those discussions.

Categories Addressed: Efficiency, Research Visibility

Daily Standup: *Start on Time, End on Time*

Practice: The daily standup starts on time and ends on time, with no exceptions.

Motivation: We must respect all attendees' time, as it is one of their most valuable commodities. One cannot manage time effectively if meetings do not follow a schedule. Dan Mezick [6] taught us that this is the single most effective practice for improving daily standup meetings.

Category Addressed: Efficiency

Daily Standup: *Parking Lot*

Practice: Discussions that are taking too long in the daily standup are placed in the "parking lot" for discussion after the standup. It is essential that the SM ensure the team remembers to review the parking lot items after the daily standup or the parking lot will have no meaning.

Motivation: Placing discussions in the parking lot is often the only way to keep the daily standup on schedule.

Category Addressed: Efficiency

Daily Standup: *Attendance*

Practice: If you are modifying the code base during the Sprint, you need to attend the daily standup every day.

Motivation: Your changes potentially affect everyone else surrounding the code. Even if you did not modify any code since the last daily standup, you need to know what impediments you may have created for the team or they have created for you, and be in a position to address them.

Categories Addressed: Part-Time Developers, Work Within Scrum.

2) Product Backlog and Writing Stories

In our Scrum team, as in many Agile teams, we use stories to capture the requirements in our product backlog. We specify that our stories must have a title, a description in the format, "As a <type of user> I want <capability> so that <business value> [7]," optional notes to provide clarity and acceptance criteria. The acceptance criteria include a point of contact that may be a stakeholder or the PO, a list of specifications and tests to be run. We also use a hierarchy of stories in our backlog, with deliverables at the top, which are broken down into epics, which are then broken down into sprint stories that are sized so the work described can be completed in a single Sprint.

In the early days of our Scrum adoption there was a perception by almost all involved that Scrum could only be used to manage code development work. It simply was not clear that we could capture other activities occurring in the group, especially those related to research. Examples include doing preliminary research, writing papers, or simply taking time to scope out solutions to a new or seemingly intractable problem. In addition, as we realized that the scope of Scrum was larger than just development, it took some time before we learned how to write good sprint stories.

A breakthrough came during a retrospective meeting when a developer stated that he felt he was not enabled to do any preliminary investigation because Scrum was only applicable to development work. In the discussion that followed it was suggested that we begin using a "design/discover meeting" story, which we have been doing ever since. This story is a small 1 to 2 story-point [5] story that describes a meeting to be held among developers to discuss the feasibility and potential solutions that govern a given research problem.

Recently, we have added a "scoping study" story that is a time-boxed [8] sprint story that is used to carve out time to investigate a problem for which too little is known to proceed further, and for which a plan of action is required. It is typically used for stubborn user support issues that need to be elevated to the level of a sprint story

In addition to our design/discovery meeting and scoping study stories we now routinely have stories that capture research work. Once we realized that Scrum could encompass research work and that the work could be described in stories, there were still challenges with respect to writing the stories. Initially, our research stories were too detailed, as we would try to eliminate any ambiguity in the description of work to be done and the acceptance criteria. Instead we found that, especially when starting a research epic, the best thing to do was to accurately capture the goals of the research in the epic from the user's point of view, but let the initial sprint stories be fairly vague. This type of story

follows Mike Cohn's suggestion that stories are, "reminders to have a conversation rather than fully detailed requirements themselves, they do not need to include all relevant details [5]." Of course, relying on conversations instead of detailed requirements requires discipline and trust. That is, instead of the PO and stakeholder overcomplicating the story they must instead trust the developers to do good work and report their results consistently at the daily standup and present a summary at the review meeting. The developers trust the PO to be available during the Sprint and to take the time to understand the outcome of the story, and to report it to stakeholders for feedback. And, most importantly, the team trusts the stakeholders to understand the results of the Sprint and help make needed course corrections.

Finally, two practices we have established that we feel are important with respect to user stories are supporting the renegotiation of user stories and the presenting and archiving the work done via done slides.

Stories: *Story Renegotiation*

Practice: The team will tell the PO as early as possible during the Sprint when a story is in danger of not being completed. This may result in the story being renegotiated. The SM will remind the team of the renegotiation possibility one-third and two-thirds of the way through the Sprint during the daily standup, emphasizing any sprint stories that seem to be struggling. The one-third point check focuses on stories that may have changed based on information discovered since the start of the Sprint. The two-thirds point check focuses on stories that are requiring more effort than expected. Renegotiation typically takes the form of modifying the acceptance criteria.

Motivation: For various reasons stories take longer to complete than expected or simply do not make sense based on information found when working on the story. This must be reported to the PO early so the PO's confidence in the team is not reduced. In addition, no team member should put the PO in a position of having to learn about or report to a stakeholder about an unfinished story at the review or planning meeting or during the following Sprint. It is much better for everyone to report delays and problems early so that there are no surprises and adjustments can be made. Note that in the latest version of *The Scrum Guide* the authors state, "During the Sprint: … Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned."

Categories Addressed: Research Visibility, Delivery.

Stories: *Done Slides*

Practice: A user story is not done until we have a written summary of the work. We call this a story's "done slide."

Motivation: We have found that having this written record to be useful for reporting on our work during review meetings and to serve as an archive of our work.

Categories Addressed: Research Visibility.

3) Planning Meeting

The manner in which the Thermal/Fluid Scrum team prepares for and executes the planning meeting is described in detail in the article *A Starting Point for Negotiations - Delivering with a Heterogeneous Team* [9]. The article describes how the PO with the help of the SM and development team use a counterintuitive Pre-Sprint Work Balancing methodology to arrive at the planning meeting with a proposed sprint backlog. This proposed backlog is sized in story points and matched to the capacity of the team using estimates of each developer's capacity for the upcoming Sprint and a preliminary matching of developers to user stories. This plan is presented to the team at the beginning of the planning meeting as a starting point for negotiations from which the actual sprint backlog is then derived in the traditional fashion outlined in *The Scrum Guide.*

Also, during the planning meeting, we use time boxing extensively. This is done when breaking a story into smaller, more achievable parts makes little sense and there are too many unknowns and/or risks for the developers to be comfortable with committing to completing the work in one Sprint. When this is done, we place a "[TB]" in the title of story so it will be immediately obvious to all familiar with our processes which stories are time boxed. We time box with story points, where the developers understand that the time box means they are committing themselves to give to the story an effort [10] commensurate with the story point estimate, and then move on to other stories in the Sprint as needed to complete the sprint backlog. Again, this is where trust comes in. The PO and Stakeholder trust that the developers will put into the story the agreed upon effort and report progress at the daily standup, review meeting and during ongoing backlog grooming. The developers trust that the PO and Stakeholders will listen to the results and modify their expectations accordingly, and any modifications will be reflected as changes in the product backlog and release plan.

Another technique that we use instead of time boxing is to soften the user story during planning by adding modifiers to the story's title, description and acceptance criteria. Adding phrases like "work towards", "start" or "begin implementing" to the title and description can make finishing a story seem much more likely. The acceptance criteria may also need to be softened by removing items or allowing partial completion, e.g. "Begin documenting options" instead of "Document options." Again, this requires trust and understanding throughout the team, puts an emphasis on communication and adds uncertainty to the planning process.

Planning: *Pre-Sprint Work Balancing*

Practice: The PO will arrive at the planning meeting with a candidate sprint backlog that follows the Pre-Sprint Work Balancing methodologies described in *A Starting Point for Negotiations - Delivering with a Heterogeneous Team*.

Motivation: Traditional Scrum planning does not adequately address the uncertainties in heterogeneous teams.

Categories Addressed: Efficiency, Delivery, Managing Uncertainty, Part-Time Developers, Specialization.

Planning: *Story-Point Estimation*

Practice: Developers do the final story point estimation.

Motivation: The people who are committing to do the work are the ones who estimate the work.

Category Addressed: Delivery

Planning: *Candidate Sprint Backlog*

Practice: The PO will arrive at the planning meeting with a prioritized list of candidate user stories for the upcoming Sprint and these stories will be "sprint ready". Sprint ready means they will have a complete description, needed notes, context and acceptance criteria. Stories can and most likely will be modified somewhat during the planning meeting, but they must be in a good state before the meeting.

Motivation: Choosing user stories from the product backlog and making stories sprint ready during the planning meeting is waste.

Category Addressed: Efficiency

Planning: *Final Sprint Backlog*

Practice: The team will only take on during the planning meeting the level of work they believe they can deliver.

Motivation: The PO must manage the backlog in such a way as to create the greatest value for the stakeholders. This requires the ability to plan with confidence for the next Sprint and beyond. This confidence can only be achieved when the PO believes that the team will deliver what they promise.

Categories Addressed: Efficiency, Delivery

Planning: *High-Value Work*

Practice: In return for the team reporting story status in a timely manner and delivering what they promise, the PO will strive to provide to the team only high-value stories that represent work that will not be unused, thrown-away, submitted for rework or otherwise wasted.

Motivation: The team must have confidence that the PO will only ask them to perform high-value work.

Categories Addressed: Efficiency, Delivery

4) Review Meeting

Our review meeting provided one of our largest challenges, and we continue to modify them in order to improve. One of the biggest problems is that our product does not lend itself to typical software demonstrations. There are no graphical user interfaces to our codes that can serve as a focal point for our demonstrations and most simulations do not run in real-time. Instead we rely on text, equations, graphs and still shots or movies of simulation results. For research and design stories, reporting on these activities to an audience of non-experts is clearly difficult.

When we first began Scrum in January of 2008 our review meetings were two-hour affairs where project status and each sprint user story was discussed in excruciating detail. Because of their length and the amount of detail given, these review meetings were not well attended by those outside of the Scrum team. There was so little overall value in the user story discussions that most team members brought laptops and only participated when the user stories they had direct interest in were addressed. Amazingly, this dreadful state of affairs continued for two years until late 2009 when Alan Shalloway and Guy Beaver of NetObjectives presented their results of a "Lean Enterprise Assessment" performed on the entire Computational Simulation Group. With respect to our review meetings, the reported stated the following: "Review should not be individuals reporting out on their activities - these should be known." and "Review should be a demonstration of completed, validated and accepted work that the team has undertaken - review time should take less than an hour." The comment was made that each discussion of a user story was like an internal status meeting for a scientific project, with too much detail for non-Scrum team members and detail that should already be known by team members because of the daily standup and other discussion occurring during the Sprint.

After this assessment we immediately set out to revamp our review meeting for our first 2010 Sprint. Despite the fact that *The Scrum Guide* states that, "This is an informal meeting", we decided to produce a slide-deck using PowerPoint for each Sprint as it provided the most convenient way to collect, display and archive our work. Though the details of the format of the slide-deck have evolved over the years, the basic one established for the first 2010 review meeting is still followed. First, we time boxed the presentation to one-hour. In the first few slides we graphically present the project status via some form of a burnup chart, a history of the team's velocity and a history of how story points have been allocated each Sprint among various funding agencies. After the status section we present the done slides depicting the work done on each sprint user story.

Challenges we have faced are the following: a) remaining on schedule, b) allowing for discussion during the presentation, c) putting the story being presented into context with respect of the team's overall deliverables and with respect to progress against the owning epic, d) providing the correct level of detail for the audience and e) attendance of stakeholders, managers and members from other Scrum teams.

a) Remaining on Schedule

After trying many things, we have found that remaining within our one-hour time box requires the developer's presenting their done slides to keep their presentations short, and the best way to do this is to let each presenter know the status of the presentation in terms of the total time remaining, the number of user stories remaining to present, and the number of slides remaining to present. Displaying the time remaining is as easy as providing an easily viewable clock. We report the number of user stories remaining by displaying a count on the various transition slides that are used to divide the presentation by project areas. Reporting the number of slides remaining is done by simply numbering each slide and noting the total slide count.

b) Allowing for Discussion

While the steps above have allowed us to present everything done during the Sprint in one hour, we found that it did not allow time for discussion. Recently, we decided that while every sprint user story needs to be archived in the slide-deck, all do not need to be presented during the review meeting. This has allowed for a much more relaxed presentation which can accommodate questions and discussion with the audience. Typically, stories that are left out are those that are one in a series of ongoing work where results are not yet

significant enough to report, or the story is only of interest to team members, and they already know the status because of daily standup meetings.

c) Putting the User Story into Context

Providing context has been one of the most difficult challenges. The most effective and simplest way is to simply have each presenter introduce the slide with a very brief description of the broader programmatic context of the slide and the status before the Sprint began. It is also useful if the presenter closes with a few sentences describing any work that remains to be done. We have tried providing context slides for each done slide, or adding dedicated context sections to the presentation, but have found the time is best spent showing the work that was done and allowing for discussions and audience questions.

d) Providing the Correct Level of Detail

Our challenge has always been to reduce the amount of detail presented. Recently we have achieved success by actually revamping the entire format of the slide-deck into what we call an "Apple-Keynote like presentation." Based on an experiment suggested in a retrospective, we decided that for our review meeting in late October 2012 we would make our presentation look like an Apple Keynote Address. We changed to a white-text on black-background slide format and eliminated everything that was not essential from our slides. We followed the suggestions given in the book *The Presentation Secrets of Steve Jobs* [11]. The book drives home the fact that for each slide audience members are asking themselves one question, "Why should I care?" and we strove to answer that question. We also relied heavily on pictures and graphics, with the first slide for almost every user story being a picture that illustrated the story's purpose. Developers were encouraged to write talking points on note cards instead of relying on slide text. Since our slide-decks also must serve as archival information, developers were told to put details into either the slide notes or in additional slides that were not shown during the meeting.

The presentation was a great success, and though we have stepped back somewhat on the strictness of the slide format, the experiment caused a step-change in how we design and deliver our presentations. We received a great amount of positive feedback on the new format and our attendance, which was already high, increased even more.

e) Attendance

Clearly, the first step to achieving good attendance is to provide a good presentation, which is what we have tried to do as described above. But even if a team can consistently deliver a good presentation, one must compete with the other items contending for a potential attendee's time. We have found two of the best ways for beating out these competitors are to simply send out a short and concise meeting reminder via e-mail the day before or the morning of the meeting and to consistently provide food and coffee.

Review Meeting: *Review Meeting Slide-Deck*

Practice: The review meeting consists of the developers, SM and PO presenting to the audience a slide-deck of relevant team statistics, such as velocity history and project status, and a subset of the sprint user stories completed via done slides. This slide-deck, with the done slides for all sprint user stories completed during the Sprint, is archived to record the work done during the Sprint and to provide a future source of data, pictures and animations.

Motivation: The review meeting is the most important avenue the Scrum team has to convey its progress and plans to a broad audience and to illicit feedback. In addition, ensuring the team statistics and each completed user story are adequately documented in the slide-deck allows the team to leverage the work done preparing for the review meeting to also create an archive of the Sprint.

Categories Addressed: Delivery, Research Visibility.

Review Meeting: *Slide-Deck Simplicity*

Practice: The slide-deck presented to the audience during the review meeting will strive to supply only the needed, salient detail, always answering the question for each audience member, "Why should I care about this story?" We must respect our audience's time and interest level, recognizing that at a high level they want to understand the context of the story, the work that was done during the Sprint, possible risks, and next steps. Finally, those running the meeting must ensure the presenters know the time remaining and amount of material left to be presented.

Motivation: We strive to have our review meetings provide value to our customers and respect the other presenters. This means that each developer will present only the salient detail relative to the time available so the meeting will flow smoothly, allow time for discussion, and ensure others will be given adequate time to speak and end on time.

Categories Addressed: Efficiency, Research Visibility.

5) Reporting to and Planning with Stakeholders

Every other Sprint, a formal review of development efforts is given to our stakeholders. We define stakeholders as those personnel who oversee the scope, schedule, and budget for our work. Stakeholder meetings include all such administrative personnel, the PO, the line manager, and relevant technical staff as needed.

During stakeholder meetings, the progress of the entire development project is reported and assessed, along with identification of near- and long-term technical risks and opportunities. Regular updates are provided demonstrating the development team's progress, and unforeseen problems are resolved as necessary. Because so much of research work is high-risk by definition, managing the technical risk is an essential part of stakeholder meetings. And because our stakeholders are invested (technically and financially) in the development work, these meetings tend to be both constructive and productive. In summary, we treat our stakeholders as equal partners in our research and development enterprise and they respond accordingly.

While the development team operates on a three-week Sprint schedule, and our stakeholder updates occur every six weeks, our formal reporting to stakeholders is carried out on a three-month quarterly schedule. This hierarchical schedule permits us to report sprint-by-sprint progress while filtering out the highs and lows of productivity that occur in each

Sprint. Our stakeholder relation processes have proven to work very well in practice using this technique. In particular, a sequence of time-boxed stories lasting several Sprints can be rolled up into a single quarterly progress report, so that hard quarterly deadlines are nearly always reached, even when some individual Sprint deadlines slip.

The fundamental principles of our stakeholder relations are those of treating our stakeholders as equal partners in our work, of candid communication and complete transparency, and of formal quarterly reporting to year-long development plans that are developed in collaboration with the stakeholder community. This approach has proven to work well for us on the full range of development tasks, ranging from low-risk routine programming of new features up to the highest-risk open-ended exploratory research efforts.

6) Retrospective Meetings

Luckily, we have never experienced significant challenges within our retrospective meetings. In fact, almost all of the practices described here arose from discussions during retrospective meetings.

Our retrospectives follow a simple format and we use a wiki page to capture and archive them. First, we look over the previous retrospective and discuss items that are flagged as either action items or experiments. This is important as retrospectives lose their value if the team does not trust that issues brought up will be addressed. Next, we move to any items located in a "Discussion Topics" section of the wiki page. Finally, we move to the answering the two questions: "What went well?" and, "What could be improved?" It is important to make it clear to all in attendance that anything affecting the team can be discussed. At first, the discussion is in an open format, which usually generates a substantial amount of discussion from which new action items and experiments are identified. Once no other issues are brought up, the SM asks each participant individually if they have anything else they would like to add. Invariably, this portion of the retrospective results in an equal amount of discussion as the general discussion, usually generated by a few people who are less comfortable talking in an open forum. Our team does not limit the retrospective's attendance; it is open to anybody who wishes to attend. We revisit this policy periodically, but it has never changed.

V. CONCLUSIONS

We have observed that managing research-heavy engineering software projects using Agile techniques appears to be little different than managing other software projects to those familiar with Agile but not research as it is performed in government laboratory, academic, or corporate research settings. For those familiar with research in these settings but not Agile, the practices used to implement Agile seem to conflict with how research is performed. From our experiences we believe that the research component of scientific and engineering software development brings with it many unique challenges, but that Agile is a very good methodology and even a natural fit for managing these projects.

In the above discussions we have identified what we believe are the unique challenges, and their causes, encountered when implementing Agile on such projects. These can be categorized into two areas. The first is overcoming a research culture that values individual achievement and fosters the reporting only of work that is finished. The second is looking past the conventional practices used to implement Agile to the fundamental principles of transparency, inspection and adaptation. The cultural norms must be overcome to achieve these principles, and the principles must be understood to successfully modify the standard practices such that they support the research and the environment in which research is normally performed.

We believe that Agile project management is woefully underutilized in the scientific and engineering research community and hope that the experiences described here will encourage others to experiment and eventually succeed.

ACKNOWLEDGMENT

This paper is dedicated to the late Sheldon R. Tieszen who in his role as the PO of the Thermal/Fluids Scrum team successfully brought Agile project management to our research-heavy engineering software team. Even though he had no experience with Agile when he began his role, Sheldon implicitly understood Agile project management and was able to implement his ideas within the Scrum framework and allowed us all to finally, consistently, deliver what we promised to our stakeholders.

We would also like to Linda Rising and the staff of Agile 2013 for their patience, assistance and invaluable suggestions while creating this paper.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] H. W. Meuer, E. Strohmaier, J. J. Dongarra, and H. D. Simon, The 41st TOP500 List, June, 2013. Available from: http://www.top500.org/list/2013/06/.

[2] K. Schwaber, Agile Project Management with Scrum, 1st ed., Microsoft Press, 2004.

[3] K. Schwaber and J. Sutherland. Scrum Guide, October, 2011. Available from: http://www.scrum.org/Scrum-Guides.

[4] P. Hawken, A. Lovins, L. Hunter Lovins, Natural Capitalism, 1st ed., Little, Brown and Company, p. 272, 1999.

[5] M. Cohn, User Stories Applied: For Agile Software Development, 1st ed., Addison-Wesley Professional, 2004.

[6] D. Mezick, http://newtechusa.net/agile/the-daily-standup/, http://newtechusa.net/dan-mezick/.

[7] M. Cohn, Agile Estimating and Planning, 1st ed., Prentice Hall, 2005.

[8] J. Rasmusson, The Agile Samurai: How Agile Masters Deliver Great Software, 1st ed., Pragmatic Bookshelf, 2010.

[9] A. Lorber and S. Tieszen, A Starting Point for Negotiations - Delivering with a Heterogeneous Team, In Agile Conference (AGILE), IEEE, pp. 148-155, 2012.

[10] M. Cohn, It's Effort, Not Complexity, Succeeding with Agile - Mike Cohn's Blog, June 21, 2010, http://www.mountaingoatsoftware.com/blog/its-effort-not-complexity.

[11] C. Gallo, The Presentation Secrets of Steve Jobs: How to Be Insanely Great in Front of Any Audience, McGraw-Hill, 1st ed, 2009.