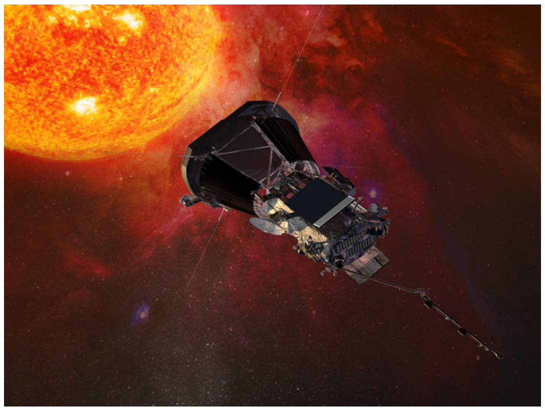
Agile Methodology for Spacecraft Ground Software Development: A Cultural Shift

***Kristin Wortman, Brian Duncan, Eric Melin**
Johns Hopkins Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723
***240-228-9634**
***Kristin.Wortman@jhuapl.edu**



***Abstract*—In the Space Exploration Sector (SES) at Johns Hopkins University Applied Physics Laboratory (JHU/APL) the development of Mission Operations Ground Software (GSW) to support NASA and Department of Defense spacecraft missions has traditionally followed the incremental build methodology. As part of our continuous process improvement effort, the Agile methodology is being introduced as an alternative approach to software development. To meet the needs of sponsor requirements and satisfy our quality management processes a tailoring of Agile is required.**

Development of ground software tools is currently in progress at JHU/APL to support NASA's Solar Probe Plus (SPP) Mission Operations. Tool development for spacecraft operation planning activities is paving the way for the cultural shift to agile methodology on candidate projects. The user community, the SPP Mission Operations team located at JHU/APL, supports the Agile Manifesto of heavy user involvement, the need for flexibility to evolve requirements and deliver frequent software releases to support readiness activities for the SPP launch in the summer of 2018. Agile was implemented following Scrum on two SPP GSW products, a spacecraft software simulator and the spacecraft activity planning tool.

The paper will briefly introduce the heritage development process and compare this process with the evolving Agile approach. The discussion will include an assessment of how Agile methodology is being tailored using Scrum to adhere to our quality management processes (e.g., peer reviews, software assurance) to meet sponsor compliance requirements for current and future JHU/APL spacecraft missions. The current SPP GSW projects implementing Agile and the experiences and lessons learned to date will be highlighted throughout the paper.

TABLE OF CONTENTS

1. INTRODUCTION

The Space Exploration Sector (SES) at Johns Hopkins University Applied Physics Laboratory (JHU/APL) located in Laurel, MD has traditionally followed the incremental build methodology for development of Ground Software (GSW) and support tools for former and presently supported space exploration missions: COmet Nucleus TOUR (CONTOUR), MErcury Surface Space Environment, Geochemistry, and Ranging (MESSENGER), New Horizons, Solar-Terrestrial Relations Observatory (STEREO) and Van Allen Probes. Currently Solar Probe

Plus (SPP) is being built by JHU/APL with a target launch date of summer 2018.

SPP is a NASA mission designed to fly into the Sun's atmosphere (or corona). The SPP mission will fly significantly closer to the Sun than any other spacecraft has done in the past. SPP will gather in situ data on the processes that heat the corona and accelerate the solar wind to solve fundamental science mysteries. The SPP science objectives are to study: coronal heating and solar wind acceleration, and the production, evolution and transport of solar energetic particles [1].

SPP is part of NASA's Living With a Star Program managed by Goddard Space Flight Center (GSFC). SPP spacecraft hardware and software subsystems are being designed and built by JHU/APL [1]. JHU/APL will also provide mission operations support for SPP.

Space exploration missions are diverse in science objectives, schedule, cost, risk and complexity. Adapting to changing economic times and staying competitive in the aerospace industry requires us to continuously seek efficient and effective ways to build high quality and reliable software products. One of the software engineering challenges of introducing an alternative approach to development in an organization is a cultural change. The other challenge is process integration to maintain compliance with sponsor requirements.

The paper will briefly discuss two SPP ground software products selected to introduce and tailor agile development to meet compliance requirements and industry standards. The software products are nearing completion and have been used to assess the acceptance of agile development by management, developers, user community and sponsors. The paper will discuss why the tailoring of agile was necessary to comply with heritage processes. The discussion will include a summary of the tailored agile framework and its implementation following Scrum. Through the SPP ground software tool development we have gained valuable insight into these areas enabling us to better understand the characteristics for candidate software products to support future space missions.

2. GROUND SOFTWARE SYSTEM

Robotic space missions such as SPP rely upon ground software to send commands to the spacecraft, receive telemetry responses from the spacecraft and evaluate data returned. The ground software provides the core command and control capability for flight operations; planning tools for spacecraft command generation and parameter management; telemetry distribution, archiving, decommutation, alarm processing and display; and assessment tools for evaluating and maintaining spacecraft performance.

On SPP we evaluated existing capabilities and determined the best path forward for command sequencing and software command sequence simulation was to develop new tools. The SPP command load generation tool (activity_planner) and software simulator (the command load constraint checker, CLCC) are being developed with modern web technologies (java, javascript, python, and mariaDB). These tools will be used frequently by mission planners to define spacecraft activity sequences and receive a quick verification that the planned sequences work as intended and do not violate mission constraints. The output of activity_planner is the input of CLCC and they will be most typically used serially.

We chose to use an agile approach with these applications because they had the following traits:

1) Significant new development (> 1 SY) with multiple (3 to 5) member teams
2) These applications provide a significant amount of user interaction. When there are multiple potential ways to interact with the system development greatly benefits from end user feedback.
3) Application end users are very active and directly available for feedback and consultation. Much of the work to specify the application behavior was undertaken by the users.
4) CLCC had many additional subsystem stakeholders as the software models contain many portions of each subsystem. Agile provides a good venue to interact with subsystem leads and help flush out constraints while they are being discovered through subsystem development.

3. CULTURAL SHIFT TO AGILE

Historically all components of our mission's ground software system were developed following the incremental build methodology. Introducing agile methodology as an alternative development approach for selected software components required a different mindset and 'buy in' from the entire team. The user community for our SPP ground software products is the Mission Operations (MOPs) team. Agile Manifesto [4] requires that the end users be embedded in the development team and assume a more participative role as compared to the end user role when following the incremental build approach. The ultimate goal of agile methodology is to define and develop the right software to accomplish the required functional objectives. Additionally, our goal for the agile approach is to provide usable, quality, reliable ground software tools in a timely manner to support pre-launch and post-launch mission operations.

The following sub-sections will highlight the fundamentals of the Incremental Build Methodology as defined at JHU/APL and the Agile Development Method implemented with Scrum. Both approaches are presented to compare the flow of life-cycle activities and the required level of team involvement to aid in understanding our decision to adopt agile for ground software development.

Incremental Build Methodology

Incremental Build Methodology begins with the traditional elicitation, definition, peer review and base line of software requirements. The mandated software requirement document is approved by the development team and stakeholders to describe the completed software product. After review and approval of the software requirements, the preliminary design commences for the fully functional software product. Once the preliminary design phase is complete, the build-specific activities (requirements analysis, detailed design, implementation, and application and system testing) commence and are completed along with the associated technical documentation for each planned major software build.

Each incremental build adds more functionality until the complete Ground Software build is delivered to Mission Operations in the final iteration. All the planned functionality defined in the approved software requirements document at the beginning of the project is delivered in the final software build. Umbrella software engineering activities (configuration management, defect tracking, peer reviews, documentation approvals) are performed throughout development for each build.

Figure 1 shows the development flow of activities when adhering to the Incremental Build Methodology. The driving factor for the multiple software build cycles is a known set of requirements followed with a preliminary design to direct sequential development activities conducted during in each planned build cycle.

Agile Development

Agile development is not a specific methodology per se, but rather a set of development frameworks that focus on highly iterative cycles, short timelines, constant feedback and prioritization of the delivered software functionality, and frequent software releases. Scrum is the most accepted Agile Development Method in industry at this time, and is the sanctioned agile approach for our candidate software products developed to support JHU/APL space missions.

Scrum is best described as a framework that can be used for managing and organizing work as a set of tasks to be completed in order to produce a working software product. However, Scrum does differ from traditional methodologies (e.g., Incremental Build Methodology, Waterfall) since it is not a standardized development process that strictly adheres to a series of steps completed in order to produce a software product. The customized structure of Scrum is the building foundation for development of the software based on values, principles and practices [2].

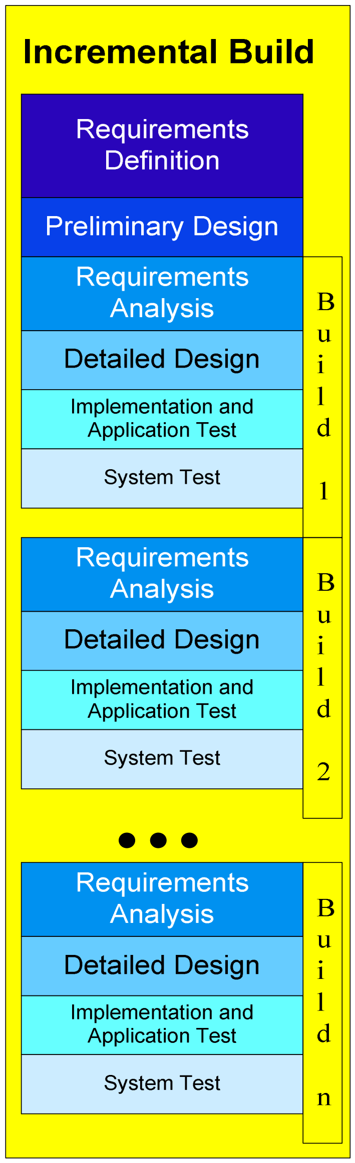


Figure 1. Incremental Build Methodology

The practices of Scrum consist of:

- defined roles for the team,
- a set of activities, and
- following a simple set of rules.

The roles are defined as Product Owner, Scrum Master and Development Team. The Product Owner represents the stakeholder community and is the customer representative who assigns the value to development tasks and establishes and prioritizes the work to be done (Product Backlog). The Scrum Master facilitates the execution of the Scrum to ensure the team follows the process and adheres to the rules of the governance agreed upon by the team. The Development Team consists of technical members from cross-disciplines who are responsible for the building and

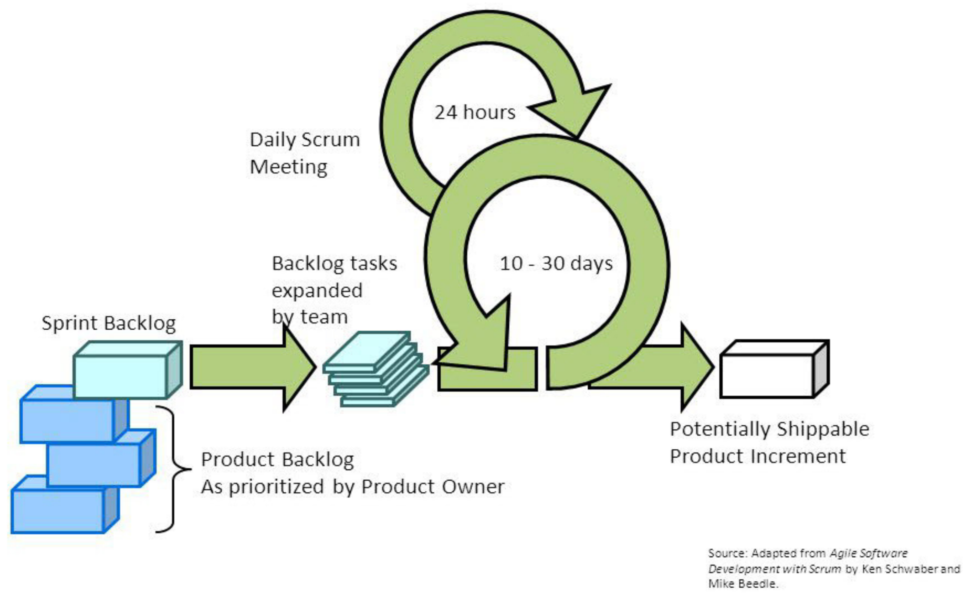

24 hours
Daily Scrum Meeting
Backlog tasks expanded by team
10 - 30 days
Sprint Backlog
Product Backlog As prioritized by Product Owner
Potentially Shippable Product Increment
Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Figure 2. Scrum Framework Representation

verification of the software product.

Activities include the iterations (or sprints), daily meetings (or scrums), execution of tasks (the sprint), retrospectives, and planning of the work (Product and Sprint Backlogs) [2]. Figure 2, Schwaber and Beedle's illustration [3] shows an accurate representation of the iterative activities performed within the Scrum framework. With Scrum, the sprints and daily scrums occur at defined time intervals.

Decision to Adopt Agile Development

Several principles defined in the Agile Manifesto prompted our decision to adopt agile for selected ground software products. Customer satisfaction through continuous delivery of usable software releases and the ability to add new functionality (requirements) throughout development [4] were key reasons for choosing the agile approach. Ground software tools such as CLCC and activity_ planner contain a highly interactive user interface component and the requirements were difficult to define early in the project since the requirements are SPP dependent and were not fully known at the onset of development.

Agile methodology stresses the importance of a co-located team environment. The benefits of end user and Product Owner continuous involvement in the decision making process is to define necessary functionality and to ensure focus is on prioritization of required functionality. Our end users (MOPs) and the developers are located at JHU/APL so frequent working sessions and face-to-face information exchange were possible.

End User Involvement

As a result of our decision to adopt the agile approach, the MOPs team was easily integrated as part of the Scrum Development Team and was actively involved in the decision making process. As integrated members of the Scrum development team for CLCC and activity_planner software, MOPs team members were required to commit to attend frequent planning and assessment working sessions and scheduled scrums. Task assignments (e.g., define report formats, define critical constraints for command sequence checker) during an iterative cycle were assigned to end users with requested completion within a short period of time (duration of a sprint).

The result of end user involvement during development has yielded critical functionality being available for assessment in early software releases of CLCC and activity_planner. The software releases were continuously evaluated by the end user and new requirements were added to the Product Backlog as needed. With agile, the lower priority functionality will naturally drop to the bottom of the list and depending on schedule and budget may never get implemented in the final software release, but this is an acceptable outcome. As we progressed through the sprints, critical functionality bubbled to the top of the priority list and was implemented in a software release in a timely manner for MOPs to begin using the software for prelaunch activities.

4. Process Compliance

Our challenge was to define agile development following the Scrum framework in an environment where adherence to industry standards (CMMi, AS9100) and compliance to sponsor requirements (NASA Software Engineering Requirements or NPR 7150.2B) is contractually required for development of space mission software products. The space department at JHU/APL has an established internal Quality Management System (QMS). The processes and procedures defined in the QMS provide a roadmap for achieving a high level of reliability and quality to meet our contractual obligations for space missions. Adhering to internal quality processes, industry standards and maintaining compliance is especially important in the development of software products that contain critical functionality to support space exploration missions with different risk classification levels.

Compliance to software engineering requirements contained within NPR 7150.2B requires specific development activities, and a hierarchy of associated technical documentation to be produced as artifacts at each development phase. However, NPR 7150.2B does not require a specific software life-cycle model be followed for development [5]. The formality of the development process and artifacts (and format) to be produced at each life-cycle phase varies between the projects depending on the assigned risk classification for the space mission. The risk classification is coupled with our internally assessed categorization of each software system and is considered for selection of the appropriate software development model for each component.

Software Assessment and Process Formality

For JHU/APL space missions we determine software development process formality based on the "tier" classification of the software product (e.g., high tier versus low tier). The "tier" or classification of the software component determines the level of process formality and rigor required for its developmental activities. Each software product is categorized into a classification according to an internal assessment of attributes that we refer to as CUES:

- **C**riticality,
- **U**se/longevity,
- **E**ffort/schedule risk, and
- **S**ize

There are two tier classifications. "High" tier is used to describe the highest level of process formality. A "high tier" process requires formal documentation, peer reviews, and independent software requirements verification. The "low tier" describes a software process that can be less formal in documentation, peer reviews and requirements verification. Overall there can be fewer development activities as part of the "low tier" process. The basis for determining if the software development effort will be categorized as "high tier" is a "yes" is given for C (criticality of the software functionality) or a "yes" is given to U (use and longevity) and E (effort, schedule and risk) and S (size of project). Otherwise, the software development effort (including those with no CUES attributes) is categorized as "low tier."

In Figure 3, the tier determination matrix depicts the CUES attributes with associated risk and other factors considered in each attribute category. Criticality and Effort/Schedule are based on risk factors (performance, environment, health, third party and legal considerations). Use/Longevity is assigned based on the characteristics of the user community and the length of the planned development and maintenance phases. The Size attribute is assigned based on the number of staff required to complete the development effort. Once the tier is selected for each software sub-system based on the CUES, it is captured in the Mission Software Development Plan along with the choice of life-cycle model to meet the requirements of the tier classification. Tier classifications can also be assigned to individual software products or components within a sub-system (e.g. CLCC, activity_planner).

We determined following the Agile methodology is most applicable to our "low tier" software products; however it could also be applicable to "high tier" products with a tailoring of Scrum to maintain compliance with our internal QMS. The two SPP Ground Software products discussed in Section 2, activity_planner and CLCC, were categorized according to the CUES and selected for development following the agile methodology with the Scrum framework. Activity_planner was classified as "Low Tier" and CLCC was classified as "High Tier". CLCC contains a critical constraint checker component to be used by Mission Operations for verification of command sequences prior to loading to the spacecraft. The requirement for the constraint checker component is mapped to a critical ground software requirement that is mapped to a higher-level SPP mission requirement.

NASA classification for a mission's software system as defined in NPR 7150.2B is also considered when assessing our software products as "high" or "low " tier. Classification ratings range from A through H with Class A-E covering engineering-related software and Class F-H covering IT and business software. Software classification is determined based on a rating scale of several factors [5]:

- Usage of the software with or within NASA.
- Criticality of the software to NASA's programs and projects.
- The extent to which humans depend on the software.
- Complexity of system development and operations.
- Development and maintenance cost of the system.

NPR 7150. 2B requirements and the amount of process rigor are tailored according to the project's software classification. Our CUES assessment in addition to the NASA software classification rating are both considered in selecting the development model appropriate for our ground software. Within the ground software, further refinement of development models can be specified for individual components. For example, our ground software contains four Computer Software Configuration Items (CSCIs): commanding and telemetry, planning, assessment and tools. Each CSCI contains multiple Computer Software Components (CSCs). The activity_planner and CLCC

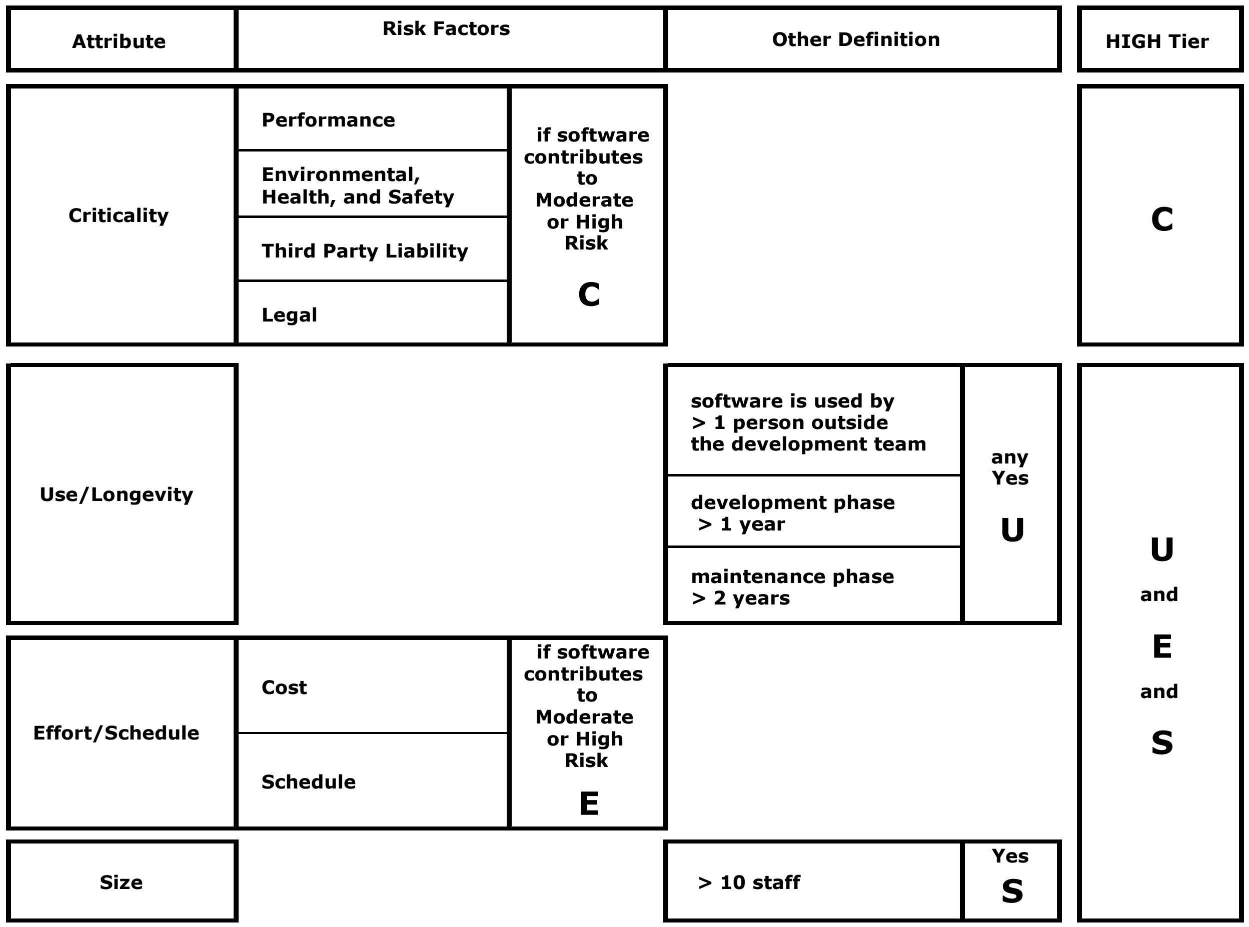

Attribute
Risk Factors
Other Definition
HIGH Tier
Criticality
Performance
Environmental, Health, and Safety
Third Party Liability
Legal
if software contributes to Moderate or High Risk
C
C
Use/Longevity
software is used by > 1 person outside the development team
development phase > 1 year
maintenance phase > 2 years
any Yes
U
U
and
E
and
S
Effort/Schedule
Cost
Schedule
if software contributes to Moderate or High Risk
E
Size
> 10 staff
Yes
S

Figure 3. CUES Categorization of Software

software are CSCs contained in the planning CSCI. The other CSCIs and planning CSCs are being developed following the traditional Incremental Build Model for SPP.

5. ADAPTING TO AGILE

As previously discussed in Section 3, there is a distinct difference between agile and the traditional approach in the requirements elicitation phase. Incremental Build Methodology requires a known set of requirements that are documented and peer reviewed and then approved at the beginning of the development cycle before proceeding to the preliminary design phase. Requirements elicitation following Scrum and the peer review process is a continuous activity throughout each iteration or development cycle. In Figure 2, Scrum has a Product Backlog and a growing list of desired prioritized functionality (requirements or user stories) procured during each iterative cycle (sprint). Planning iterations prioritize the Product Backlog and the desired set of requirements (Sprint Backlog) to be implemented in the next iteration.

Documentation following the Scrum approach is considered as evolving rather than complete until the final software release. Evolving documentation is prevalent during the requirements, design and test phases as new functionality is being added to the Product Backlog. The method of documentation can also be less formal than the traditional methodology. For example, a formal tool (e.g., IBM Rational DOORS) may or may not be used for documentation. Less formal media may be used instead for documentation of requirements, design and user documentation: spreadsheets, Word documents, PowerPoint slides or a Wiki.

Traditional methodologies follow a defined process with a set of required artifacts subject to peer review and approval before proceeding to the next phase. Our QMS requires peer reviews and software assurance oversight be conducted on the development process and its resultant products. Artifacts still exist with Scrum but they are created in an iterative way and can take on a different form from iteration to iteration. With Scrum the software assurance engineer has been integrated into the development team and becomes an active participant in the working sessions. Working sessions consist of a series of planning and review (or demonstration) sprints. Periodic retrospective meetings may also be held to capture lessons learned and to assess what is going well and where changes need to be made for continuous process improvement moving forward.

Tailoring of Scrum

Scrum was tailored where it made sense in our agile development process to include requirements to adhere to our internal processes as defined in QMS for both "High Tier" and "Low Tier" products. Tailoring the Agile process included adding requirements for Scrum to define:

- level of formality of peer reviews,
- required technical documentation,
- periodic baselines of software products,
- configuration items and management,
- inclusion of software assurance engineer as a member of the development team,
- need for independent software requirements verification, and
- inclusion of additional requirements for safety critical software.

6. CONCLUSION

The experience with SPP ground software (CLCC and activity_planner) has enabled us to move forward in defining agile as an alternative development model for candidate software to support space missions at JHU/APL. We have successfully defined the "best fit" agile process to be fully compliant with our internal quality management system by embedding the appropriate requirements according to the heritage "High Tier" or "Low Tier" CUES assessment of the software.

Several factors have contributed to the success of following an agile approach for CLCC and activity_planner software development. A strong commitment was acquired from management and developers to "buy in" to agile and follow the defined set of rules of Scrum. Our end users assumed a participative role on the Development Team which differed from their role on past missions following the traditional approach. The acceptance of Scrum included the realization that not all functionality would be implemented in the final software release as lower priority tasks may not get addressed due to schedule and budget constraints.

Embedding the end users in the development team has resulted in continuous training paralleled with verification and acceptance of each software release. End user involvement in the decision making process and prioritization of functional tasks yielded customized deliveries of working software by the Scrum team. The frequent incremental software releases supported the functionality in a timely manner and is supporting SPP pre-launch mission operations activities.

Another advantage of agile was software defects were continuously being uncovered by all members of the Scrum Development Team throughout development. End users exercised all software releases to expose deficiencies early during development which were prioritized and resolved quickly. Early end user involvement was instrumental in the quality design of a reliable user interface through continuous feedback on usability gathered during demonstrations. In addition, the end users were being trained as the software matured.

Our experience to date with adopting the Agile Development Model implemented with Scrum has resulted in definition of an alternate process to produce quality and reliable ground software tools delivered at frequent intervals. We have identified characteristics for candidate ground software products to support future space missions. We also concluded that our CUES assessed "High Tier" categorization of software requires careful evaluation before choosing the Agile Development Model, but is achievable if using a tailored process. When considering going agile in any organization some tailoring may be required in order to maintain compliance with internal quality processes and to meet compliance requirements levied by sponsors.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the sponsorship of NASA's Science Mission Directorate – Heliophysics Division and the Living with a Star Program Office at the Goddard Space Flight Center.

REFERENCES

[1] Johns Hopkins University, Applied Physics Laboratory Solar Probe Plus web site: http://solarprobe.jhuapl.edu/

[2] Kenneth S. Rubin; Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley Professional, July 26, 2012.

[3] K. Schwaber, M. Beedle; Agile Software Development with Scrum, Pearson, October 28, 2001.

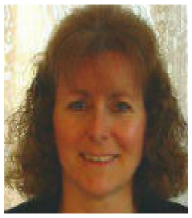
[4] K. Beck, J. Grenning, R. C. Martin, "Twelve Principles of Agile Software", Manifesto for Agile Software Development, http://agilemanifesto.org.

[5] "NASA Software Engineering Requirements," NPR 7150.2B, National Aeronautics and Space Administration, Washington, D.C., Appendix D (November 19, 2014).

Biography

Brian Duncan *received a B.S. in Computer Engineering from Carnegie Mellon University in 1989, and an M.S. in Computer Science from Johns Hopkins University in 1993. Brian has worked at JHU/APL since 2006, and is the Supervisor of the Space Systems Implementation Branch. He works with multi-disciplinary teams developing Space projects for NASA and DoD sponsors. With a deep passion for software development methodologies, Brian brings over twenty years of experience in the commercial and government sector leading highly efficient software teams in the full product life-cycle. Having helped two very different cultural organizations adopt Agile Development methods, Brian understands the necessary steps for technical training, social engineering, and project adoption at all levels of the company hierarchy.*

Eric Melin *is currently the Ground Software Lead for the Solar Probe Plus mission at the Johns Hopkins University Applied Physics Laboratory. He was the Ground Software Lead for the New Horizons and MESSENGER missions. Eric also developed ground software for the Van Allen Probes mission. Before coming to JHU/APL in 2008, Eric architected financial exchange trading systems with Capital Markets Consulting. He was a Technical Director at OPNET Technologies responsible for enhancements to a modeling and simulation tool of military data networks. Eric has a BA in Electrical Engineering from Cornell University (1999) and a MEng in Computer Science from Cornell University (2000).*


Kristin Wortman *received a B.S. in Computer and Information Science from University of Maryland University College, in 1990, an M.S. in Software Engineering, a joint degree from University of Maryland and University of Maryland University College in 2000. She has been with JHU/APL since 2007. Currently she is the Solar Probe Plus Spacecraft Software Assurance Engineer. Previously she was the software acceptance test lead for Solar Probe Plus and Van Allen Probes. Prior to joining JHU/APL senior professional staff, she was a senior computer scientist for Computer Sciences Corporation (CSC) for two tenures totally twenty-two years. She supported a team of NASA Goddard scientists and developed instrument flight and data analysis software for the STEREO observatory. When not working for CSC she was employed by several software companies as a developer for another twelve years supporting data processing and analysis software development to support various NASA missions. She is also an adjunct associate professor for University of Maryland University College in the Computer and Mathematical Sciences Department Undergraduate Studies since January 2001.*