Challenges of Applying Conventional Software System Safety to Agile Software Development Programs

Melissa A. Emery; APT Research; Huntsville, Alabama, USA

David B. West, CSP, P.E., CHMM; Science Applications International Corporation; Huntsville, Alabama, USA

Abstract

Modern systems are increasingly dependent on software for status monitoring, control, and safety. Software system safety originally evolved at a time when large software development followed a "waterfall" approach in which requirements definition, architecture design, coding, test, and deployment were conducted sequentially. This allowed time early in the program for completing several conventional system safety tasks, such as the system safety program plan (SSPP) and various hazard analyses. Many modern software development programs follow the Agile development approach, in which "sprints" are conducted, to rapidly and incrementally define the architecture, write code, and test it, often before all system requirements are established. Agile development does not align well with the conventional system safety approach. System safety engineers applying conventional software safety will find that by the time they complete and obtain approval of the SSPP, developers have already completed several sprints. To better support modern development programs, a modified software safety approach should be developed to allow software contributions to hazards (causes and controls) to be identified and assessed, before the development team has completed the software architecture and design. This will ensure software hazard causes are adequately mitigated and safety significant software adheres to the level of rigor requirements.

Introduction

Software safety is a fairly modern subdiscipline of system safety. The importance of performing software safety analyses to identify, assess, control, and accept software contributors to safety risk has continued to rise as modern systems have become more and more sophisticated, automated, and interconnected with other related or supporting systems. System automation and/or control frequently involves multiple sensors throughout the system and transfer of data or commands among various subsystems or control nodes. Proper system functioning obviously depends on all applicable data transfers and command executions being done correctly, at the right time, and in the proper sequence. System functions must also be controlled such that they do not continue indefinitely. To facilitate the development of error-free software in increasingly complex and safety-significant systems, software safety processes have been established to ensure thorough capturing of all software requirements, careful design of the software architecture, strict adherence to coding standards, and thorough testing to ensure any software errors are identified and eliminated.

Software development processes themselves have also been evolving significantly in recent years. As discussed in the next section, many software development efforts in the past followed a "waterfall" approach, which was similar in many regards to the approach used for overall system development. The waterfall approach featured distinct software lifecycle phases that were each conducted sequentially – that is, each was completed in turn before transitioning to the next phase. For instance, a thorough effort to identify and articulate all software requirements was embarked upon at the beginning of the program. It was only after the requirements were essentially completed that the team would move on to the architecture design phase. Similarly, the software implementation, or coding, was not begun until the architecture was considered complete. And finally, the system would be tested with the entire suite of software when all the coding was complete. Conventional software safety processes, also described in the next section of this paper, are more suited to the waterfall software development approach.

Complexities of modern systems, and their software, have led to a newer approach known as the Agile software development method. The Agile method is characterized by a more rapid succession of discrete efforts, known as "sprints," each of which is focused on developing only a portion of the software at a time. A more thorough overview of the Agile method is provided in the section of this paper that follows the next section.

The rapid evolution of software development from the conventional waterfall approach to the Agile method has led to some situations where project teams have tried to apply conventional software safety processes to an Agile software development effort. Conventional software safety processes are not well suited to the Agile software development method. In the time required for the software safety analyst to prepare a thorough and comprehensive SSPP, obtain

customer review comments, incorporate comments, and route a final SSPP for approval signatures, the development team will likely have completed multiple sprints. These and other challenges are described in more detail in the section below, titled "Issues with System/Software Safety Processes and Agile SW Development."

Recommendations for improved software safety processes, to make them better align with the Agile software development method, are provided in the section below titled "Suggested Changes to System/Software Safety Process for Agile Development."

Typical Software Development and Software Safety Process

In recent years, the Safety community made a shift from conventional system safety processes to a methodology that includes software safety. This is a result of systems relying more on software and less on physical robustness. Gone are the days of triple redundant computer systems with independent hardware/software backups. Today's space vehicles and soldier-toting battlefield technology does not allow for that degree of physical redundancy. Even though software has become extremely complex in its use, isolation by redundant or independent computing systems cannot always be guaranteed. Although software has always been part of the system in System Safety, due to its complexity it has been shown to require additional attention. The need for extra focus on software contributions and mitigation to system hazards sparked the need for the DoD Joint Software System Safety Engineering Handbook (JSSSEH) and commercial documents like DO 178C (Software Considerations for Airborne Systems and Equipment Certification). The conventional system/software safety process falls in line with earlier methods of software development. The waterfall approach, for example, allows for the safety process to start and end in parallel with the system process. See Figure 1. The Agile process however, does not coincide with the system/software safety process at all. The specific issues are discussed following a brief overview of the Agile software development process.

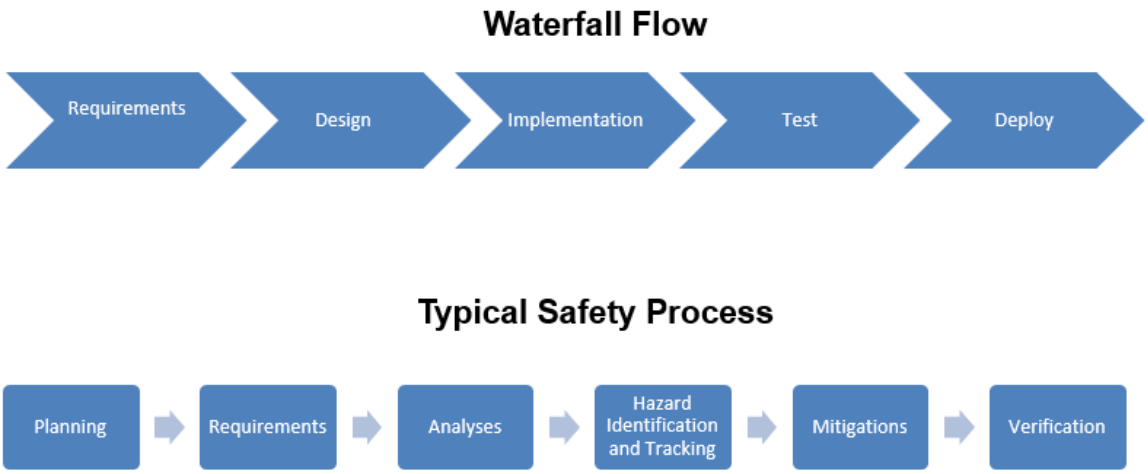

Figure 1 — Waterfall and Safety Processes

Overview of Agile Software Development Process

The Agile Development methodology is rooted in various movements that sought to reduce waste in manufacturing and production. The Toyota Production System, developed between 1948 and 1975, was a precursor to what has become known as "lean manufacturing" (ref. 1). Lean manufacturing, in turn, emphasizes many aspects of efficient and effective production that have been incorporated in Agile Development. In February 2001, a concerned group that called themselves the Agile Alliance met at Snowbird, Utah and authored the *Manifesto for Agile Software Development*, in response to the need they perceived for "an alternative to documentation driven, heavyweight

software development processes" (ref. 2). The Agile Manifesto articulated four key values that were expressed as preferences for certain aspects of software development over others (ref. 3):

1. Individuals and interactions over process and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

Beyond this statement of key values, the Agile Manifesto listed the guiding principles shown in Table 1 below (ref. 4).

Table 1 — Guiding Principles of Agile Development

No.	Principle
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4	Business people and developers must work together daily throughout the project.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity--the art of maximizing the amount of work not done--is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

In Agile Development, the software architecture is initially designed with only enough detail to allow the development team to begin their work; it is subsequently refined during the development process. This reflects an understanding that the requirements can evolve as the program matures, and the architecture must be allowed some flexibility for adjustment as the requirements change. Stated another way, since the main gauge of progress in Agile development is a working system (Principle #7 above) and the development of functionality is done incrementally, architecture is done incrementally to support the development. With this flexibility of the architecture comes the need for interfaces between system components to be loosely-coupled, so that the interfaces can be adjusted as necessary, if and when the architecture is modified during development.

Another important aspect of Agile Development is that since the architecture is incrementally designed, documentation should also be written incrementally. Documentation of the architecture in Agile Development is viewed not so much as a measure of project completeness, but rather as a means to facilitate communication between the customer and development team.

In Agile Development, individual requirements are expressed as a series of "user stories." These user stories are used both in planning the development work to be done, and as a gauge of how much of the work has been accomplished. Development work is done in a succession of discrete work efforts of constant length, referred to as "sprints." The duration of sprints is typically set at somewhere from 1-4 weeks. As outlined in reference 5, the process for planning and conducting each sprint includes:

- Selection of the customer's highest priority user stories
- Estimation of user story points for selected user stories
- Identification of tasks for each user story, and estimation of effort by developers

- Elaboration of user stories, design, code, and test
- Demonstration of a working system
- A retrospective and improvement of the process

In each sprint, the customer reviews the system and can redirect the priorities for the next sprint. Progress is tracked by noting the total number of user stories completed. The development team adheres to a discipline of never slipping a release date, but instead, slipping one or more user stories, if necessary.

Issues with System/Software Safety Processes and Agile SW Development

Agile Development is a marked departure from the waterfall process described earlier. In the waterfall, or Software Development Life Cycle (SDLC) process, it would not be uncommon for the developers to meet with the customer and spend a matter of months in a painstaking, back-and-forth exchange to reach agreement on a comprehensive set of requirements, thoroughly documented, before development work is begun. The pace and work sequence of this approach was well suited to the conventional system safety effort in which a major emphasis was on sequential completion of safety documents in a certain order.

The primary issue, or "disconnect," between conventional system/software safety processes and the Agile Development method is the difference in mindset between, on one hand, the Agile method that acknowledges the need for flexibility and the likelihood of course corrections along the way, and, on the other hand, the conventional system/software safety approach that generally expects a single effort to draft, get comments on, and finalize a given safety document, and when that document is complete, moving on to the next (with the same mindset).

As noted in Figure 1 above, when conventional system safety methods were followed, planning and requirements definition were completed first, before proceeding with analyses, hazard identification, and hazard tracking. System safety planning and requirements definition are accomplished through publishing of the System Safety Program Plan (SSPP). Much as the waterfall software development methodology placed emphasis on a protracted effort to fully develop and document the software requirements before beginning work on development of the code, conventional system safety would typically devote all its resources at the beginning of the project to preparing the SSPP, subjecting it to a thorough review and comment period by the customer, incorporating customer comments, and finally routing it for signatures by multiple parties. This process could easily take more than a month.

After completion and approval of the SSPP in a conventional system safety program, the first of the hazard analyses – often a functional hazard analysis (FHA) – is begun. Not only can this analysis and capturing of its results in a formal deliverable document take even longer than preparation and approval of the SSPP, it has the added burden or complication of requiring participation by potentially multiple members of the development team – most or all of whom it will be difficult, if not impossible, for the team to make available to support the system safety effort, because they are fully committed to the development effort.

By the time the system safety analysts are done preparing and obtaining approval on the SSPP, as well as completing what might be only the first of several analyses to identify and assess hazards, the development team will probably have planned and completed multiple sprint cycles, likely without the benefit of System Safety's participation or input. System safety analysts are often left with a feeling of being well behind on the project before they get started. See Figure 2 below.

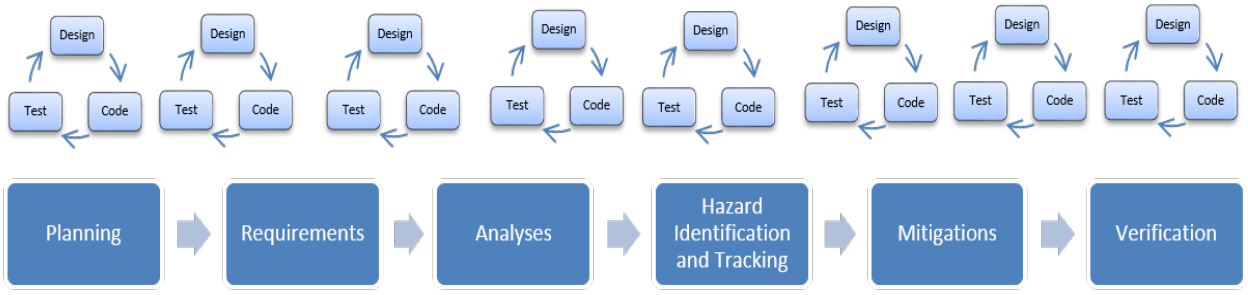

Figure 2 — Relative Time-Phasing of Agile Development Sprint Cycles with Conventional System/Software Safety

Suggested Changes to System/Software Safety Process for Agile Development

As more software development programs shift in the Agile development process direction, it is time for system/software safety to also make a shift. For Agile development programs, it seems necessary to tailor the safety process. It is recommended to start with generic safety processes and let them grow and expand with the Agile processes. This is illustrated in Figure 3 below.

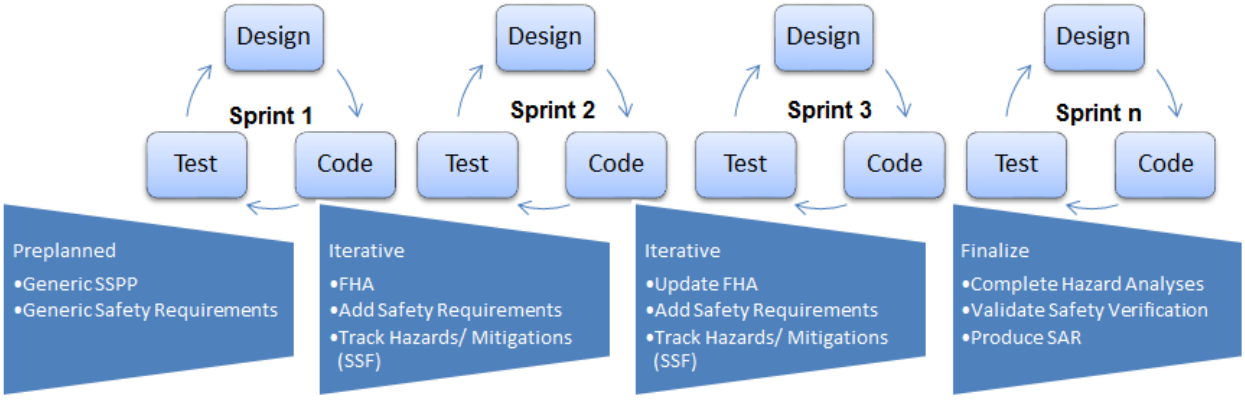

Figure 3 — Tailored System/Software Safety Process for Agile Development

The conventional system safety process should be tailored to fit the progress of Agile methods. Recommendations include the exclusion of some safety analyses (i.e., Preliminary Hazard Analysis (PHA), Fault Tree Analysis (FTA)) and focusing on expanding generic analyses and processes to encompass the progression of the system. Specifically, the following suggestions are put forth for Agile software development projects:

1. **Use a Standard System Safety Program Plan (or none—just refer to MIL-STD-882).** Don't spend unnecessary time creating this document. Review MIL-STD-882E and add any additional specifications or regulations to the Statement of Work (SOW).
2. **Impose Generic Safety Requirements at the Onset.** This sets the tone for safety's involvement. It helps developers keep safety in the decision process. Allocate generic safety requirements for Design, Code, and Test activities (see JSSSEH for suggestions). For example, for Design, impose safe state, memory, integrity, and fault detection type requirements. For Code activities requirements about dead/unused code, storage, and other coding standards should be imposed. For Test activities, document the safety testing and

level of rigor (LOR) testing required so these can be accomplished in the unit level testing and planned for in the system testing.

3. **Initiate a Functional Hazard Analysis (FHA) at the Onset.** This provides management and guidance of safety questions with respect to degraded operation, loss of operation, etc. This allows the analysis to grow with the project instead of waiting for other typical analyses (e.g. PHA) and system documentation to be available. If time permits, conduct a PHA later in the cycle. The FHA will aid in generation of safety significant functions and requirements which should be modified, if required, throughout the process.
4. **Conduct Several Requirements Analyses.** For Agile Development, it is more productive to perform several requirements analyses in contrast to using the final program documentation. Augmenting with change impact analysis ensures the requirements are constantly evaluated and documented in the program and safety artifacts.
5. **Attend Sprint Meetings to Understand Content, Changes, Future Expectations, etc.** After each Sprint meeting, reevaluate the FHA and generic safety requirements. Implement the necessary changes and review with System Safety Working Group (SSWG) members on a regular basis.

It would be necessary to evaluate and expand the safety requirements throughout the sprint cycles. Specific project/program safety requirements should be added to the generic requirements, as required, and to the program documentation (i.e., system specification). The FHA should evolve into a system hazard analysis. The hazards should be identified and tracked as required by MIL-STD-882E. The safety engineering support to an Agile Development system should still be guided by the conventional foundation; however, it is best supported by a change in the sequence of events and the types of safety analyses conducted.

Summary

Experiencing firsthand Agile development systems has led the authors to determine that a modified safety process is needed for software following this type of development. If certain safety products are pre-set at the beginning of the Agile process, and modified later if required, software safety can be proactive rather than in a constant reactive state. Generic safety requirements and program planning is the key to allowing an Agile and effective safety process. In addition to preplanned requirements and analysis, modification and sequence of safety events/analysis is recommended to ensure software safety provides the analyses and assessments necessary to identify the safety-significant software in terms of hazard causes and hazard mitigations. These suggested modifications to the system safety process should be implemented whenever Agile software development is expected.

References

1. Toyota Production System (TPS) & Lean: A Brief Overview. Accessed June 20, 2016. http://www.strategosinc.com/toyota_production.htm.

2. History: The Agile Manifesto. Accessed June 20, 2016. http://www.agilemanifesto.org/history.html.

3. Agile Software Development: A Tour of its Origins and Authors. Accessed June 20, 2016. http://www.ibm.com/developerworks/rational/library/mar07/pollice/

4. Principles Behind the Agile Manifesto. Accessed June 20, 2016. http://agilemanifesto.org/principles.html.

5. Moon, Roger. “Enabling Agility in SED’s Systems Engineering Process with Agile Development” (briefing charts used for presentation during quarterly review to U.S. Army/AMRDEC/Software Engineering Directorate). October 9, 2012.

6. U.S. Department of Defense. MIL-STD-882, Revision E. “Standard Practice for System Safety.” May 11, 2012.

7. U.S. Department of Defense, Joint Software Systems Safety Engineering Workgroup. “Joint Software System Safety Engineering Handbook (JSSSEH).” August 27, 2010.

Biographies

Melissa A. Emery, Senior Safety Engineer, APT Research, 4950 Research Drive, Huntsville, Alabama 35805, USA, telephone – (256) 327-3396, facsimile – (256)837-7786, e-mail – memery@apt-research.com.

Melissa Emery holds a BS in Mathematics from the University of Houston, Clear Lake. She began her career working software change requests, pre-STS-1, at Rockwell International in Downey, California. She moved to Houston, Texas in 1986 and supported Space Shuttle Avionics software change activities involving Space Shuttle Mission Operations. In 1991 she moved to Huntsville Alabama and transitioned to software safety supporting the Safety and Mission Assurance (S&MA) Office at the NASA Marshall Space Flight Center (MSFC). Melissa has been employed at APT Research Inc. since 2002. Her career spans 20 years of system/software safety experience supporting NASA and the U.S. Army. She also has experience in software Airworthiness and has taught several system safety and software courses while at APT Research. Melissa has been active in the Tennessee Valley chapter and she served as the International System Safety Society (ISSS) Secretary and Executive Vice President. She is currently serving the ISSS in the capacity of Director of Member Services.

David B. West, CSP, P.E., CHMM, Chief Systems Engineer, Science Applications International Corporation (SAIC), 6723 Odyssey Drive, MS 21, Huntsville, Alabama 35806, USA, telephone – (256) 971-6494, facsimile – (256) 971-7314, e-mail – david.b.west@saic.com.

David West earned a BS in Nuclear Engineering from the University of Cincinnati in 1984. He has over 27 years of service with SAIC, where he provides system safety engineering expertise for various defense projects at the U.S. Army's Redstone Arsenal. He is currently the President of the Tennessee Valley Chapter of the International System Safety Society. For the past 6 years, Mr. West has chaired the G-48 System Safety Committee, which is organized under the Systems, Standards, and Technology Council of SAE International. From 2008 through 2013, Mr. West served on the Board of Directors of the Board of Certified Safety Professionals.